KERNEL EXTENDED REAL-VALUED NEGATIVE SELECTION ALGORITHM (KERNSA)

THESIS

Brett Andrew Smith, Civ

AFIT-ENG-13-J-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-13-J-07

KERNEL EXTENDED REAL-VALUED NEGATIVE SELECTION ALGORITHM (KERNSA)

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

Brett Andrew Smith, B.S.

Civ

June 2013

AFIT-ENG-13-J-07

KERNEL EXTENDED REAL-VALUED NEGATIVE SELECTION ALGORITHM (KERNSA)

Brett Andrew Smith, B.S.
Civ

Approved:

Dr. Gilbert L. Peterson, PhD (Chairman)	14 May 2013 Date
Lt Col Jeffrey D. Clark, PhD (Member)	14 May 2013 Date
Dr. Gary B. Lamont, PhD (Member)	14 MAY 2013 Date

AFIT-ENG-13-J-07

Abstract

Artificial Immune Systems (AISs) are a type of statistical Machine Learning (ML) algorithm based on the Biological Immune System (BIS) applied to classification problems. Inspired by increased performance in other ML algorithms when combined with kernel methods, this research explores using kernel methods as the distance measure for a specific AIS algorithm, the Real-valued Negative Selection Algorithm (RNSA). This research also demonstrates that the hard binary decision from the traditional RNSA can be relaxed to a continuous output, while maintaining the ability to map back to the original RNSA decision boundary if necessary. Continuous output is used in this research to generate Receiver Operating Characteristic (ROC) curves and calculate Area Under Curves (AUCs), but can also be used as a basis of classification confidence or probability. The resulting Kernel Extended Real-valued Negative Selection Algorithm (KERNSA) offers performance improvements over a comparable RNSA implementation. Using the Sigmoid kernel in KERNSA seems particularly well suited (in terms of performance) to four out of the eighteen domains tested.

To God, who made this work possible and who is always there for me.

Table of Contents

Page

List of Figures

List of Tables

Table Page

List of Symbols

Symbol		Page
$\mathcal{X}$	the input space	7
n	number of dimensions in $\mathcal{X}$, $n = \|\mathcal{X}\|$	7
$\mathcal{H}$	the feature space	20
ϕ	the mapping function $\phi : \mathcal{X} \to \mathcal{H}$	20
$\langle x, x' \rangle$	scalar (or dot) product between x and x'	21
$k(x, x')$	dot product in feature space	21
$\|\|x\|\|$	the length of x	22
d	number of detectors	35
v	number of supporting vectors	35

List of Acronyms

Acronym	Definition
AIRS	Artificial Immune Recognition System 14
AIS	Artificial Immune System 63
ANN	Artificial Neural Network 18
ANOVA	Analysis of Variance 57
AUC	Area Under Curve 63
BIS	Biological Immune System 5
FNR	False Negative Rate 10
FPR	False Positive Rate 33
IDS	Intrusion Detection System 5
KERNSA	Kernel Extended Real-valued Negative Selection Algorithm 63
ML	Machine Learning 63
NSA	Negative Selection Algorithm 27
PCA	principal component analysis 18
RBF	Radial Basis Function 27
RNSA	Real-valued Negative Selection Algorithm 63
ROC	Receiver Operating Characteristic 63
SVM	Support Vector Machine 18
TNR	True Negative Rate 10
TPR	True Positive Rate 33
UCI	University of California, Irvine 47
WDBC	Wisconsin Diagnostic Breast Cancer 59

KERNEL EXTENDED REAL-VALUED NEGATIVE SELECTION ALGORITHM (KERNSA)

I. Introduction

The Artificial Immune System (AIS) is a type of statistical Machine Learning (ML) algorithm based on the Biological Immune System (BIS) that is applied to problem solving [16, 18]. Example applications of the AIS include anomaly detection [30, 31, 45], network Intrusion Detection Systems (IDSs) [48, 60], clustering [19, 81], and optimization [13, 29].

One-class classification (also known as anomaly detection) is used to classify data when only a single class of data are available or the other classes are not common [76]. The data available are considered "normal" and future data may be considered anomalous if they are not sufficiently similar to the normal data. For example, healthy human cells display a certain set of characteristics which differ when the cell becomes cancerous or otherwise sick; the healthy cells could be given to a one-class classifier in order to help detect cells that need treatment.

The Real-valued Negative Selection Algorithm (RNSA) is a type of AIS algorithm used to classify samples in one-class problem domains [16, 76]. This research combines kernel methods into the RNSA. The combination of kernel methods with other existing ML algorithms has been shown to increase the performance of those algorithms [11, 21, 47]. This chapter introduces the motivation, goals, and assumptions of the work in this thesis investigation.

1.1 Motivation

The RNSA AIS uses an "affinity function" (defined as Euclidian distance in many RNSA implementations) to measure the difference between a detector and a sample in the input space. By using the Euclidian distance as the affinity function and a constant detection distance, the RNSA forms hyper-spherical detectors in the input space. Enhancements to the shape representation of detectors is identified as an area of future research by Dasgupta [14]. By having differently shaped detectors, there are more options for choosing a detector shape that more closely fits a given problem domain. Indeed, the affinity function and detector shape need to be tailored to the problem domain [27].

1.2 Goals and Assumptions

This thesis investigation proposes the Kernel Extended Real-valued Negative Selection Algorithm (KERNSA), which uses a given kernel function as the affinity function. Using a kernel function as the affinity function, along with other kernel-related modifications to the RNSA, can change the shape of the resulting detector in the input space.

A new ML algorithm will never outperform existing algorithms in all problem domains [40, 84], but a new algorithm may perform better than an existing one on average in many problem domains. The decision of what ML algorithm to use on a given problem will always be dependent on the problem and the characteristics of the data. However, by adding kernel methods to the RNSA, detectors use a variety of detection shapes that are able to fit the given problem domain better than the traditional hyperspheres employed by the RNSA. An improvement by enhancing the detection shape's representation was anticipated by Dasgupta [14].

Additionally, this research seeks to modify the RNSA's classification output from a binary classification to a continuous value which can be used as class membership probability. This value is used to create Receiver Operating Characteristic (ROC) curves in this research, but is also a general enhancement to the RNSA for future work.

This thesis examines several questions in regard to KERNSA:

- Does KERNSA offer improvements over the traditional RNSA?
  - Are these improvements statistically significant?
  - How does KERNSA compare to other AIS algorithms?

The assumptions made in this research include the following:

- All features in the problem domains are real values.
- The datasets chosen are representative of the larger set of one-class problem domains.

1.3 Result Overview

In this research, testing was performed on the diabetes, iris, ionosphere, sonar, Wisconsin Diagnostic Breast Cancer (WDBC), and wine datasets from the University of California, Irvine (UCI) ML dataset repository [1] and the comb, intersection-thin, pentagram-mid, and ring-thick datasets from Dasgupta's [15] one-class dataset repository. Comparing classifiers on the basis of Area Under Curve (AUC) using these datasets shows that KERNSA offers a statistically significant performance improvement over a comparable RNSA implementation with every dataset on at least one of the dataset's labels. Using the Sigmoid kernel in KERNSA seems particularly well suited (with a significantly higher AUC) to a subset of the datasets tested.

KERNSA is compared to other AIS algorithms and found to perform well, though it only ranks first in accuracy in one of the six datasets compared. That KERNSA only places first in one of these datasets may be explained in part because the classifiers were tuned to AUC, another performance measure, instead of classification accuracy. A more direct comparison is left to future work.

1.4 Thesis Overview

This chapter introduces the research, goals, and limitations of the work in this thesis. Chapter 2 discusses background topics and research. Chapter 3 proposes the design of KERNSA and the experiments used in this thesis. Chapter 4 presents the results of these tests and analyzes them. Finally, Chapter 5 provides a conclusion of this research and details areas of future work.

II. Background

The application of kernel methods (as defined in Section 2.3) to pre-existing Machine Learning (ML) algorithms has resulted in significant improvements to the original algorithm, particularly with kernel clustering [11, 21, 47]. Because of these improvements, this work explores combining kernel methods with the Artificial Immune System (AIS). The AIS is a form of ML that is based on the Biological Immune System (BIS). The Negative Selection Algorithm (NSA) is a type of AIS based on the biological T-cell [12, 14] and is a supervised learning algorithm used in one-class classification. This thesis presents a modification of the NSA based on prior work combining kernel methods with clustering.

This chapter introduces the background knowledge necessary to discuss the modifications proposed later in this investigation. First, ML, the AIS, and their relevance today are reviewed. Then, the specific AIS algorithm this research modifies is discussed. Some objections against the AIS are examined. Next, kernel methods, as used in this investigation's modification, are considered. Finally, the chapter concludes by reviewing previous work on kernels and AISs.

2.1 Machine Learning (ML)

Mitchell [56] describes ML as, "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." ML makes possible tasks such as determining whether an email is relevant for the user [62], detecting credit card fraud [57], network Intrusion Detection Systems (IDSs) [73], or facial recognition [32], without requiring a human to make all of the decisions. ML is also useful because users do not need to provide explicit domain knowledge to the learning algorithm, which is highly

desirable since domain knowledge is not always easily expressed or well understood by domain experts [68].

There are three broad areas of ML [68]:

- In *supervised learning*, the learning algorithm generates a classifier which maps inputs to outputs [68]. During the learning phase, the learning algorithm receives input (a sample) that has already been mapped to the appropriate output label (usually by a human) so that the classifier can predict future unlabeled input. This is also known as inductive inference, observing a phenomenon and generalizing it [40]. Figure 2.1 presents a visualization of the general supervised learning process. Labeled training data are presented to the learning algorithm. The learning algorithm then creates a classifier which is capable of predicting the label of future input.

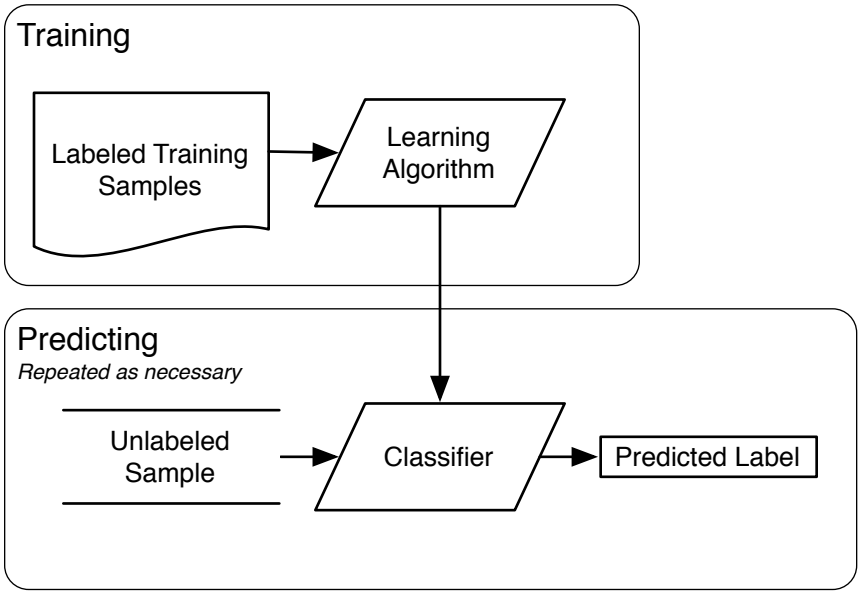


Figure 2.1: The supervised learning process.

- In *unsupervised learning*, the learning algorithm is given no hints about the structure of the input (e.g., the labels in supervised learning) [68]. Unsupervised learners try to

discover possible structuring of data. The most common use of unsupervised learning is clustering [68].

- *Reinforcement learning* is a method of creating intelligent agents through a system of rewards and punishments without specifying how the task will be accomplished [46]. An example of reinforcement learning is a rat taught what button to push; when the rat pushes the correct button it gets a treat, but it receives a shock when it pushes the wrong one. The rat quickly learns to do what will be rewarded instead of punished.

Formally, $\mathcal{X}$ is the input space with $n = |\mathcal{X}|$. Each dimension in $\mathcal{X}$ is called a *feature*. Each $\mathbf{x} \in \mathcal{X}$ where $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is a sample drawn from $\mathcal{X}$ independently according to a fixed but unknown distribution [40]. A label y can be given to every vector $\mathbf{x}$ according to a fixed but unknown conditional distribution. A labeled sample is often formally represented as $[\mathbf{x}, y]$ in ML literature.

Finally, every learning algorithm draws from a class of hypothesis functions $\mathcal{F}$ [40]. The hypothesis function space $\mathcal{F}$ represents every classifier that could be created by a learning algorithm given any arbitrary set of training samples. This simply means that the classifiers are limited by the $\mathcal{F}$ they are drawing from; if the learning algorithm separates classes linearly in the input space, it won't do well on data that are not linearly separable. As an alternate way of viewing this problem, learning algorithms are sometimes seen as searching the space $\mathcal{F}$ for the best hypothesis for the given training samples [23].

During the review of the literature, it was noted that ML classifiers fall into one of the three following classification types:

- One-class classification is interested in recognition of a single class as opposed to discriminating between classes [41]. For example, a one-class classifier can be used when there are samples of apples available for training and it is necessary to recognize whether unknown physical objects are apples [79]. This type of classification is particularly useful when there is a "normal" state that is desired. Examples of "normal"

states include healthy, non-cancerous human cells or motor operations (amount of noise, vibration, exhaust, etc). The term "one-class classification" comes from Moya and Hush [58], but it is also referred to as outlier/anomaly detection [67], novelty detection [7], concept learning [41], or data domain description [78].

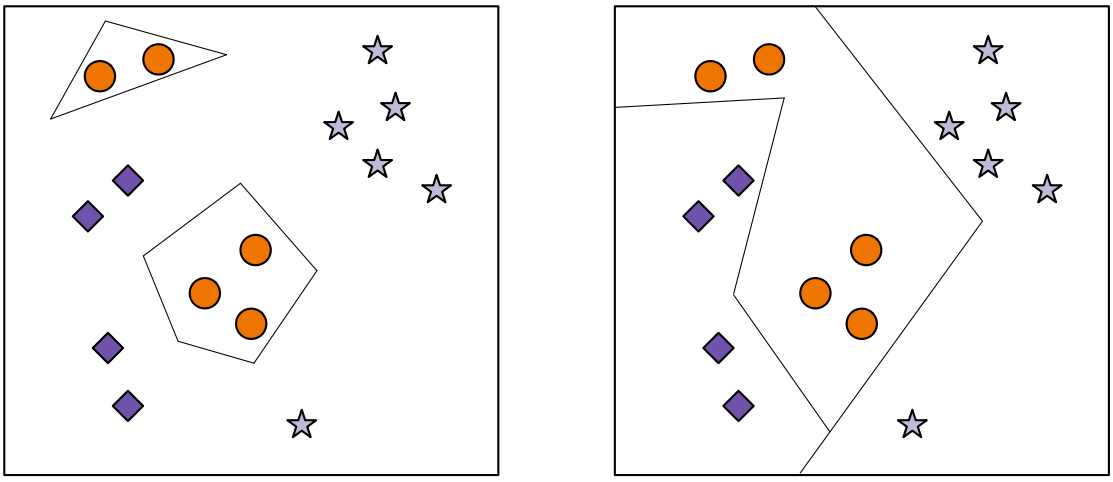
- Binary classification discriminates between two classes [41, 71]. For example, given samples of apples and pears and the knowledge that an unknown physical object is either an apple or a pear, one can create a classifier to discriminate the likely class of the unknown object [71].

- Multi-class classification applies when the classifier can discriminate between more than two classes (eg, a pear, apple, or orange) [71]. A multi-class classifier can always be created by combining one-class or binary classifiers using various techniques [71].

A visualization of the differences between recognition (one-class classification) and discrimination (binary and multi-class classification) is shown in Figure 2.2. Figure 2.2a is a general recognizer for the circles in this training set. Any new, unlabeled sample that falls outside the boundaries shown in Figure 2.2a would be predicted as "not a circle" by a one-class classifier. Figure 2.2b is discriminating between the three classes, any new samples would be predicted to be the same as the other known samples sharing the compartment of the new sample by a multi-class classifier. These figures are not illustrating any particular learning algorithm.

2.1.1 Classification Performance.

An important balance in ML is creating a classifier that does not overfit or underfit the underlying problem [40]. An overfit occurs when a classifier fits the samples it was trained on very well, but has an overly complex decision boundary and does not generalize well to future samples; an overfitting classifier thus loses predictive power [40]. However,



(a) Recognition of circles.

(b) Discrimination of the three shapes.

Figure 2.2: Recognition vs discrimination.

a classifier can also underfit (or generalize) too much such that the classifier's decision boundary is much simpler than the true boundary the classifier is attempting to find [40].

Some measure of a classifier's performance is used to help determine how well the classifier will do on future input [40, 53]. Although estimating a classifier's performance cannot directly identify if the classifier is under- or overfitting, it can assist in this determination since a side effect of under- or overfitting will usually be lower performance. Additionally, though a learning algorithm could create a classifier that assigns labels at random, this classifier would not be of much practical interest, and a good performance measure should be able to identify this relatively useless classifier.

Accuracy is often used to measure the performance of a classifier, and is sometimes the only evaluation criteria given in ML research [53]. The total accuracy of a classifier is defined as the number of correct predictions it makes over the total number of predictions [40].

A more detailed representation of performance (for a one-class or binary classifier) is reporting the number of true positives, true negatives, false positives, false negatives, and

their associated rates, True Positive Rate (TPR), False Positive Rate (FPR), True Negative Rate (TNR), and False Negative Rate (FNR). In AIS literature [35, 45, 76], a *positive* sample is one that has been classified non-self (regardless of the actual class), and a *negative* sample has been classified as self. A *true* sample is one that has been correctly classified, while a *false* one has not. So, we have the following definitions [40].

$$\begin{aligned}
\text{TP} &= \text{number of true positives (correctly classified as non-self)} \\
\text{TN} &= \text{number of true negatives (correctly classified as self)} \\
\text{FP} &= \text{number of false positives (classified as non-self, actually self)} \\
\text{FN} &= \text{number of false negatives (classified as self, actually non-self)} \\
\text{TPR} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\
\text{FPR} &= \frac{\text{FP}}{\text{FP} + \text{TN}} \\
\text{TNR} &= 1 - \text{FPR} \\
\text{FNR} &= 1 - \text{TPR} \\
\text{Accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}
\end{aligned}$$

Since many statistical ML classifiers output a real valued measure of a sample's class membership, it is possible to obtain various TPRs and FPRs for different threshold values on this output value. For example, if a larger output from a classifier indicates greater probability that the sample is self, a threshold value of negative infinity results in a TPR of 0.0 and a FPR of 0.0 (everything is classified as self) while a threshold value of positive infinity results in a TPR of 1.0 and a FPR of 1.0 (everything is classified as non-self). These results form the *Receiver Operating Characteristic (ROC) curve* [40], which places the FPR on the x-axis and the TPR on the y-axis. An example ROC curve is presented in Figure 2.3. Note that a curve from a classifier that falls below the diagonal line is worse than random guessing. One of the advantages of reporting the TPR and FPR and presenting the ROC curve is that they are not sensitive to prior class distributions, unlike accuracy [40].

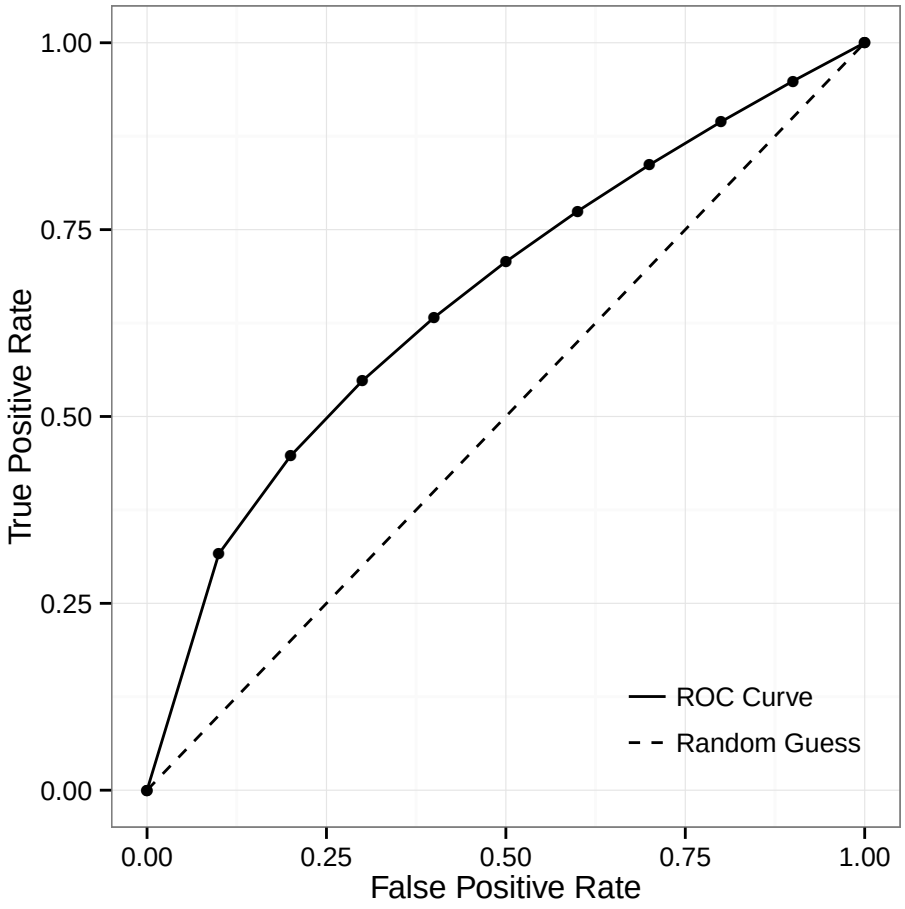


Figure 2.3: Example ROC curve.

The performance of a classifier can be represented as the Area Under Curve (AUC) of the ROC curve [40]. Since the ROC curve is not sensitive to class distributions, the AUC is not either [8]. The AUC is also preferred because it is a more consistent and discriminating measure than accuracy [53]. Thus, the AUC is preferred to overall accuracy when giving a *single number* evaluation or comparison of machine learning classifiers [8, 52]. AUC is a strong choice regardless, since it has been shown that classifiers that are optimized for AUC also perform well for accuracy [52, 53]. Even though some concerns in using

the AUC measure have been raised [34, 54], no other alternative single measure is widely recommended at this time [54].

Several different methods of calculating the AUC have been suggested [40]. The most popular of these is to create the ROC curve and then take the area under that curve [40]. Since the ROC curve is bounded in a 1×1 square, an AUC of 1.0 represents perfect predictions, 0.5 represents the performance one would expect from random guessing, and 0.0 represents a classifier that gets every classification incorrect (though interestingly, one could simply reverse the predictions for a perfect classifier). This research calculates the area using the trapezoidal rule (the points on the curve are finite).

Though AUC is recommended and used in this research for the reasons given, it is worth noting that there are always trade-offs in choosing a performance measure [40]. Modifying an algorithm to increase performance in one measure may very well decrease performance in another measure, and the performance measure of greatest interest will depend on the use of the algorithm [40]. When ranking several classifiers, separate performance measures rank the classifiers quite differently [40]. As such, performance measures should be chosen carefully depending on the needs of the problem domain [40]. AUC is by no means a panacea of performance measurements, but it is a good choice for the needs of this research.

There are many other methods of calculating performance that have not been mentioned, see Japkowicz and Shah [40] for more. Other performance measures include: error rate, Cohen's kappa, F measure, root-mean-square deviation, and KL divergence [40].

Since the classifier needs to generalize to avoid overfitting, samples other than those used in the training must be used in the testing/estimation process. Testing a classifier with samples that were used during training gives a performance estimate that is very optimistic [83]. This optimism occurs because one is usually more interested in knowing how a classifier will perform with samples it has never seen before rather than how well a classifier can repeat verbatim what it was trained upon.

2.1.2 Cross Validation.

Performing k-fold cross validation takes full advantage of all available samples for both training and testing [68]. In k-fold cross-validation, the dataset order is randomized and then split into k equally-sized mutually exclusive folds, each fold being a subset of the original dataset. A total of k rounds of learning are performed, each round uses a different one of the k folds as the test fold and uses the remaining folds for training. The final averaged accuracy from the k rounds will be a good estimate of the accuracy of the learning algorithm on this dataset [49, 68]. Figure 2.4 illustrates the first two rounds of a 5-fold cross validation.

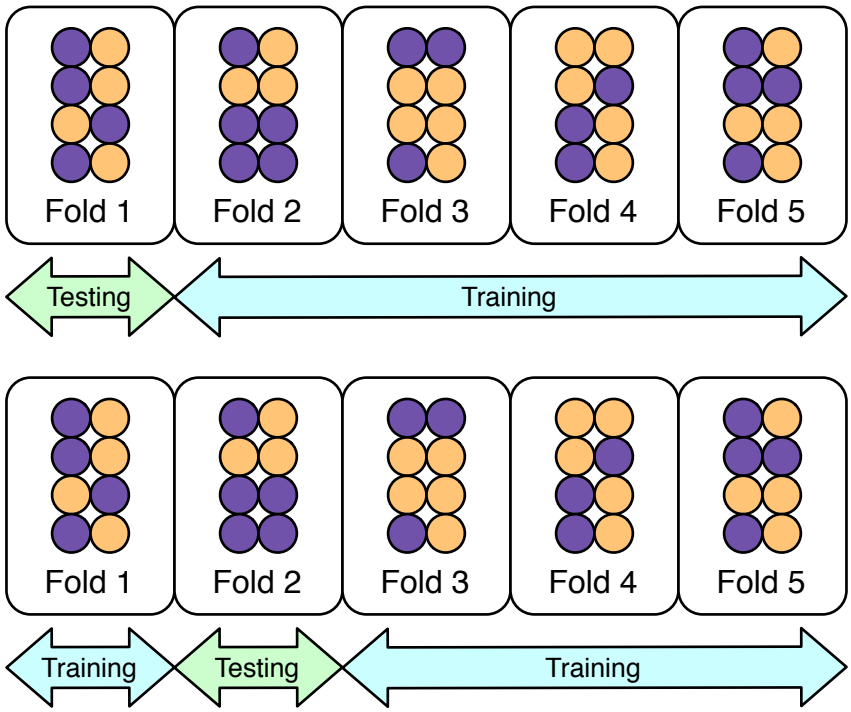


Figure 2.4: Two iterations of five-fold cross-validation. The circles represent samples, the colors in the circles represent the sample's label.

Common values of k are five or ten [40, 68]. It has been shown that $k \approx 10$ achieves the best performance estimates [83]. Higher values of k give an estimate that is more likely

to be closer to the true value, but takes approximately k times longer computationally than doing a single training and testing [68].

In order to ensure that each fold is representative of the original dataset, the folds are often stratified. That is, each fold is composed of approximately the same percentage of labels as the overall dataset [40]. Overall, stratification gives a less biased estimate of the true accuracy [49].

2.2 Artificial Immune System (AIS)

The AIS is a form of ML based on the BIS [14, 18]. The AIS takes on many different forms depending on which form of biological immunity it is borrowing from. The most discussed forms of the AIS include the NSA, the immune network model, and clonal selection [14, 37].

- The NSA is based on the biological T-cell [12, 14] and was first introduced by Forrest et al. [25] in their discussion of self and non-self. The target concept of the NSA is the complement of the self samples used to train the classifier [16]. In other words, the NSA tries to learn what features any possible non-self sample could have in the given problem domain. Since *only* self samples are used to train the NSA, it is considered a one-class classifier [16]. This thesis investigation uses a form of the NSA known as the Real-valued Negative Selection Algorithm (RNSA) (discussed in Section 2.2.1).

- The immune network model is based on the biological B-cell [14]. This model operates on the concept of B-cells stimulating and suppressing each other to identify antigens. During training, the network is presented with samples of both classes of data. The cells in this model are cloned and mutated during training upon successful recognition of classes [14]. These cloned cells are then integrated into the network at the nearest B-cell to the clone, or removed if it is unable to integrate. The

Artificial Immune Recognition System (AIRS) is one example of an immune network algorithm [82].

- Clonal selection is the basic theory that biological immunity cells, which recognize antigens, are cloned (with mutations) and proliferated within the host, thus being *selected* over immunity cells that are not as effective [14]. CLONALG, introduced by De Castro and Von Zuben [17], implements these principles.

2.2.1 Real-valued Negative Selection Algorithm (RNSA).

The RNSA is the NSA applied to a real-valued input space. The RNSA randomly generates *detectors* in the input space. These detectors are n-dimensional shapes that cover regions of the input space. Any point that is within the detector's covered area is considered a match for the detector. After being randomly generated, detectors that match any given example of self are discarded, which is similar to how T-cells function in the BIS. An unlabeled, future sample is considered non-self if any detectors match it. This process is shown in Figure 2.5. Detectors can be any shape or combination of shapes in the input space. Though many RNSA implementations use hyperspheres as the detector [77], hyper-ellipsoids [72], hyperrectangles [60], hyper-steinmetzs [6], and a combination of simple shapes [24] have also been explored.

The concept of shape space was introduced by Perelson and Oster [63]. Shape space is a subset of the input space. That is, shape space is a finite n-dimensional shape with hyper-volume V in which all possible self and non-self samples for a given problem domain exist. Having a finite hyper-volume allows the RNSA to function. Otherwise, the RNSA would be attempting to cover an infinite volume with finite detectors. Perelson and Oster also defined *affinity* within the shape space which is a measure of similarity between two points within the space [63]. Euclidian distance is usually used as the space's affinity function [77]. To stay consistent with other ML literature, this research refers to the shape space as the input space.

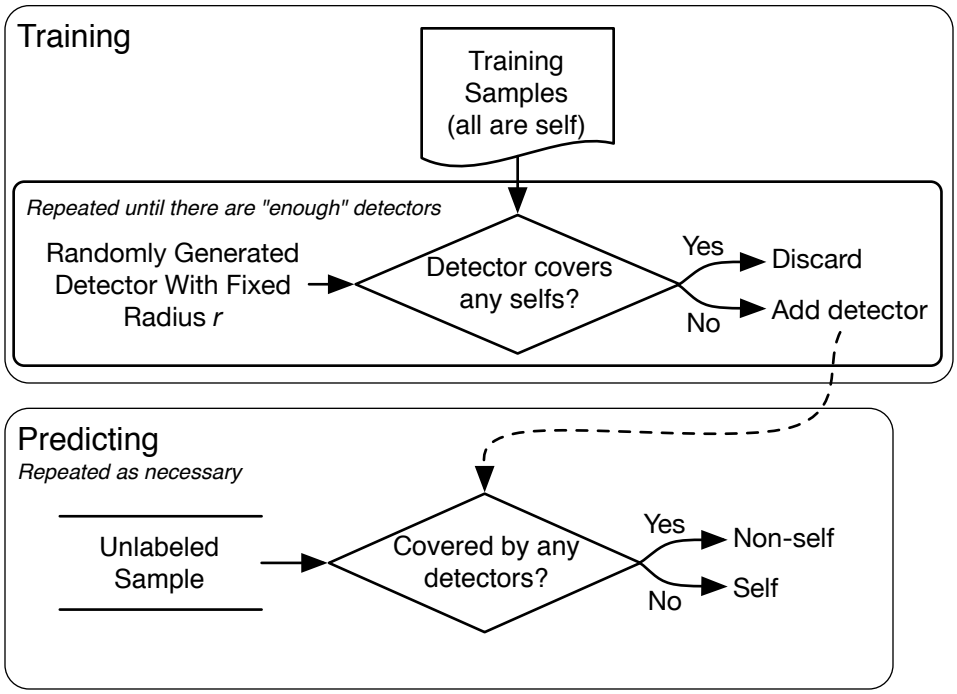


Figure 2.5: The Negative Selection Algorithm (NSA).

An example two dimensional input space with fixed radius circular detectors can be seen in Figure 2.6. Note that areas where the detectors overlap with self would result in false positives, while areas inside the shape space that is neither self nor covered by a detector would result in false negatives. Finally, note that only the centers of the detectors need to be inside the shape-space in the RNSA.

Ji and Dasgupta [42] have proposed an alternative RNSA known as *V-detectors* (variably-sized detectors). This algorithm generates a set of random points in the input space and uses each as the center of a new detector's hypersphere [16]. If a point is within a fixed distance of the self samples or is covered by existing detectors, it is removed [16]. Otherwise, the detector's radius is set to the distance to the nearest self sample [16]. Detector generation stops when the algorithm estimates that a certain amount of input space coverage has been achieved [16].

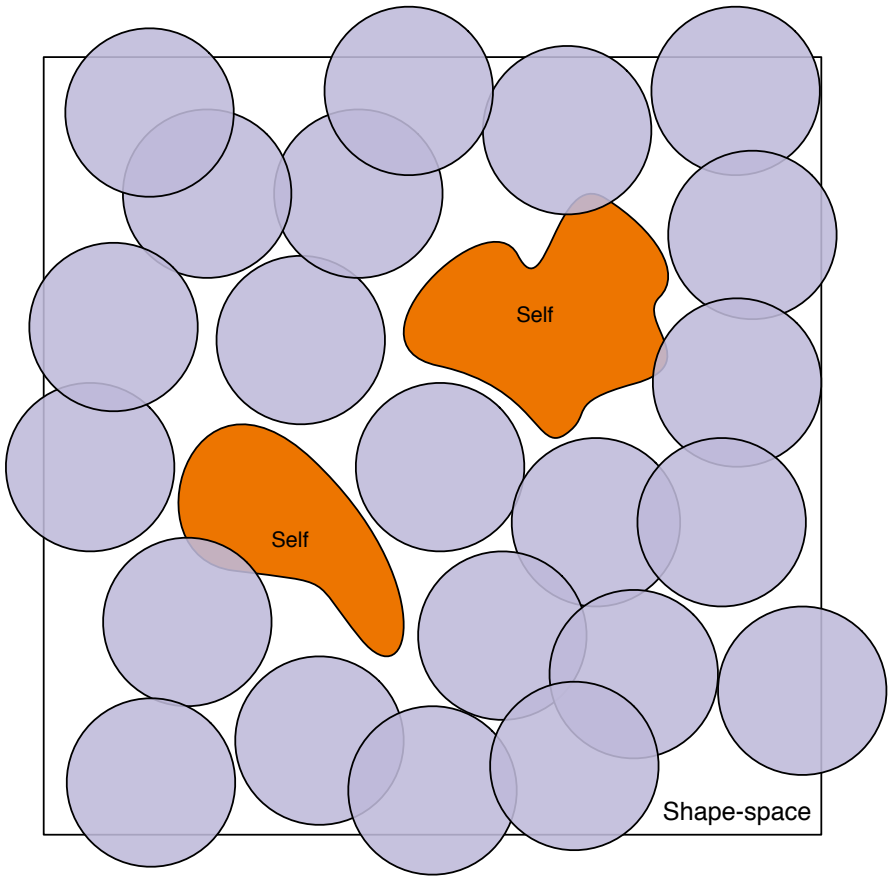


Figure 2.6: Illustration of self and non-self regions with detectors.

2.2.2 Objections to AIS and RNSA.

There are several objections that have been raised by researchers questioning if the RNSA is an appropriate ML technique. These objections include the objection of dimensionality [77], of representation [43], of randomness and coverage [76], of the usefulness of negative selection [74], of overfitting and oversearching [26], and the limitation of one-class classification [43, 75].

The objection of dimensionality. As the number of dimensions of the input space increases, various computational phenomena arise. For example, the hyper-volume of a fixed-radius hypersphere goes to zero as the number of dimensions approaches infinity [77].

Conversely, the hyper-volume of the input space increases exponentially as the number of dimensions approaches infinity [77]. This "curse of dimensionality" is universally agreed upon as an important issue within the AIS community [16, 43, 77] and in ML at large [3, 28, 38]. The chosen detector shape becomes a very important factor in the RNSA as the number of dimensions increases [6].

It is important to remember that the curse of dimensionality is not limited to the AIS, but affects ML as a whole. There are ML techniques used to decrease the number of dimensions such as principal component analysis (PCA) or feature selection. However, the utility of these in the NSA is dubious since the NSA is trained only on self data but is interested in recognizing self versus non-self. Information that PCA or feature selection discard may be of critical value in determining what is non-self in the future.

The objection of representation. The RNSA operates on a continuous, real-value input space, but some data are discrete. For example, if one feature is states in the United States of America, is the affinity between Utah and Texas greater than or less than that between Ohio and Iowa [43]?

This is another issue that affects many real valued ML algorithms. In particular, these questions must be answered when representing data for Artificial Neural Networks (ANNs) or Support Vector Machines (SVMs). Other ML algorithms such as decision trees can work with discrete data natively.

One solution is to break each possible discrete value into a separate binary feature indicating the presence or absence of the value. However, this increases the dimensionality of the input space and still assigns arbitrary distances to discrete information. Another solution is to place the discrete values in $\mathcal{X}$ equally spaced from each other, which is arbitrary in both order and spacing but does not increase dimensionality. This research does not use data that contain discrete features and thus avoids this issue altogether.

The objection of randomness and coverage. Since the NSA is a random process, confidence in the detector's estimated input space coverage is necessary to minimize variance in the generated classifier's accuracy [76]. Estimating the input space covered by detectors results in a significantly higher time complexity for detector generation.

It is important to remember that there are four sources of variation in a learning algorithm's accuracy estimation: the variation of the selected test data, the variation of the selected training data, the internal randomness of the learning algorithm, and the random classification error from mislabeled samples [22]. These variations are problematic for most ML algorithms, but since the NSA relies heavily on internal randomness through detector generation it is a much larger source of concern.

V-detectors have been proposed as a partial solution to this problem [42]. For this research, this problem is considered an acceptable risk and left for future work.

The objection of the usefulness of negative selection. The goal of the RNSA is to fill the negative space (the space not belonging to self) with detectors. Stibor and Timmis [74] argue that positive selection is the better approach to this problem. That is, each self sample in the training set acts as a detector for future self samples. This approach also removes the randomness of the training algorithm.

Using a positive selection algorithm is certainly the best choice for some problem domains: it is a simple algorithm that may be able to completely separate labels using this straightforward method. However, the RNSA will still be the better choice for other problem domains (see the "no free lunch" theorem [84]). The RNSA may be able to map the self versus non-self space more accurately than the positive selection algorithm under certain conditions such as varying self sample densities or self spaces with complex shapes.

The objection of overfitting and oversearching. The NSA does not have any mechanism to prevent overfitting or oversearching [26]. Overfitting occurs when the learning algorithm tries to match the training samples too closely and is unable to generalize well to

novel data. Oversearching is similar and occurs when a hypothesis is chosen that matches chance relationships in the training samples that are unlikely to occur in general.

These problems are largely left up to individual implementations to solve. Good accuracy estimation helps in discovering the proper tweaking to avoid overfitting and oversearching. Using a sufficient number of training examples for a problem also decreases the chances of overfitting or oversearching.

The limitation of one-class classification. Training on only one class suffers the loss of additional information which may be useful in improving accuracy [43, 75]. But, one-class classifiers are a real need [69] and anomalous samples may be unavailable for training. Alternatively, anomalous samples may exist but be vastly outnumbered by self samples or there may be uncertainty which samples are anomalous in a large set of samples.

2.3 Kernel Methods

It has been suggested that the affinity function used in the NSA needs to be tailored to the problem domain [27], yet there has not been much research into what affinity functions work well with the NSA (excepting the different shape representations mentioned in Section 2.2.1). This paper presents using kernel methods as the affinity function. Kernel methods are a class of largely interchangeable similarity functions often used in other areas of statistical ML, particularly with SVMs.

ML algorithms often look for a way to linearly separate one class of samples from another. Sometimes this simply is not possible given the arrangement of the samples in the input space. In order to find a better linear separation, it is often helpful to transform the samples from the input space ($\mathcal{X}$) to a higher dimensional space known as the feature space ($\mathcal{H}$) and *then* find linear separation. Formally, samples are transformed into $\mathcal{H}$ using a mapping function $\phi : \mathcal{X} \rightarrow \mathcal{H}$. An example of some points mapped from a two dimensional input space to a three dimensional feature space via some ϕ is visualized in Figure 2.7.

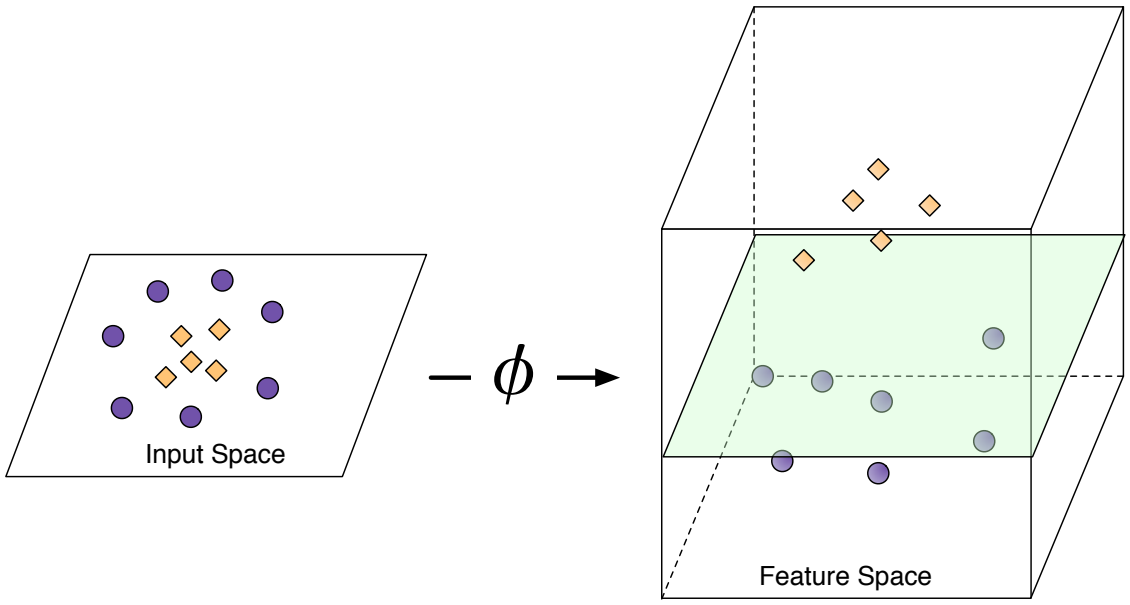


Figure 2.7: An example of a possible input space to feature space transformation.

Computing the space transformation for every sample would be computationally expensive, especially if $\mathcal{H}$ has many dimensions. Instead, just knowing the dot product between any two samples in feature space is usually enough, since the dot product allows the generalization of several operations like projection and the distances or angles between samples [33]. The dot product between two vectors x and x' is represented by $\langle x, x' \rangle$. A *kernel function* is defined as $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$. To avoid working in the potentially high-dimensional space $\mathcal{H}$, one can choose the feature space $\mathcal{H}$ such that the dot product can be computed directly using a pair of samples in the input space $\mathcal{X}$ [70]. Indeed, it is not even necessary to explicitly specify the mapping function ϕ [47]; this is known as the *kernel trick* [68, 70, 71]. The kernel trick reduces the computational complexity associated with using space transformations in ML.

Some of the more commonly used kernels are listed in Table 2.1. A *Radial Basis Function (RBF) kernel* is a kernel function that can be expressed in the form $k(x, x') = f(d(x, x'))$ where d is a metric on $\mathcal{X}$ and f is a function on $\mathbb{R}_0^+$ [71]. These RBF kernels

exhibit some interesting properties. In particular, if the metric function being used is distance (as in the Gaussian and Inverse Multiquadratic kernels), then the kernel functions are translation invariant [71]. In other words, if it is known that $||x - x'|| = 2$ (where $||x||$ is the length of x), the result of the Gaussian or Inverse Multiquadratic kernels can be computed regardless of where x and x' are in $\mathcal{X}$.

Table 2.1: A few commonly used kernels.

Kernel Name	**Definition**	**Notes**
Linear	$\langle x, x' \rangle$	With this kernel, $\mathcal{H} = \mathcal{X}$. That is, no transformation is applied.
Gaussian [71]	$e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$	$\sigma \in \mathbb{R}$. Sometimes σ is instead represented as λ such that $\lambda = \frac{1}{2\sigma^2}$ and $\lambda \in \mathbb{R}^+$. This is a RBF kernel.
Inverse Multiquadratic [33]	$\frac{1}{\sqrt{\|x - x'\|^2 + \sigma^2}}$	$\sigma \in \mathbb{R}$. This is also a RBF kernel.
Sigmoid [51]	$\tanh(a\langle x, x' \rangle + c)$	$a, c \in \mathbb{R}$.

To calculate distance in $\mathcal{H}$ between two points [11, 70], let p and q be points in $\mathcal{X}$. The distance is calculated as,

$$D_k(p, q) = k(p, p) + k(q, q) - 2k(p, q). \tag{2.1}$$

To calculate distance between the centroid of a set of points and a single point in $\mathcal{H}$ [47, 64], let p be a point in $\mathcal{X}$ and let Q be a set of points in $\mathcal{X}$. Let n_Q be the number of points in the set Q. The distance in $\mathcal{H}$ between p and the centroid in $\mathcal{H}$ of Q is

$$D_k(p, Q) = \sqrt{k(p, p) + \frac{1}{n_Q^2} \sum_{q \in Q} \sum_{q' \in Q} k(q, q') - \frac{2}{n_Q} \sum_{q \in Q} k(p, q)}. \tag{2.2}$$

2.3.1 Kernel Clustering.

Clustering is a form of unsupervised learning that attempts to solve the problem of discerning multiple categories in a collection of objects on the basis of those objects alone [68]. Clustering is helpful as a form of data exploration [5]. Some examples of clustering include the clustering of gene information to discover possible similar functionality [4] or clustering exoplanets and stars to determine similarity and create a taxonomy for naming different types [68].

Many researchers have explored applying kernel methods to clustering and have shown promising results [11, 21, 47]. Kernel clustering modifies traditional clustering algorithms by using kernel methods for the distance measurements instead of the traditional linear distance [11]. Because of the relative success from kernel clustering, this research applies kernel methods to the RNSA to see if a similar gain can be achieved with an AIS.

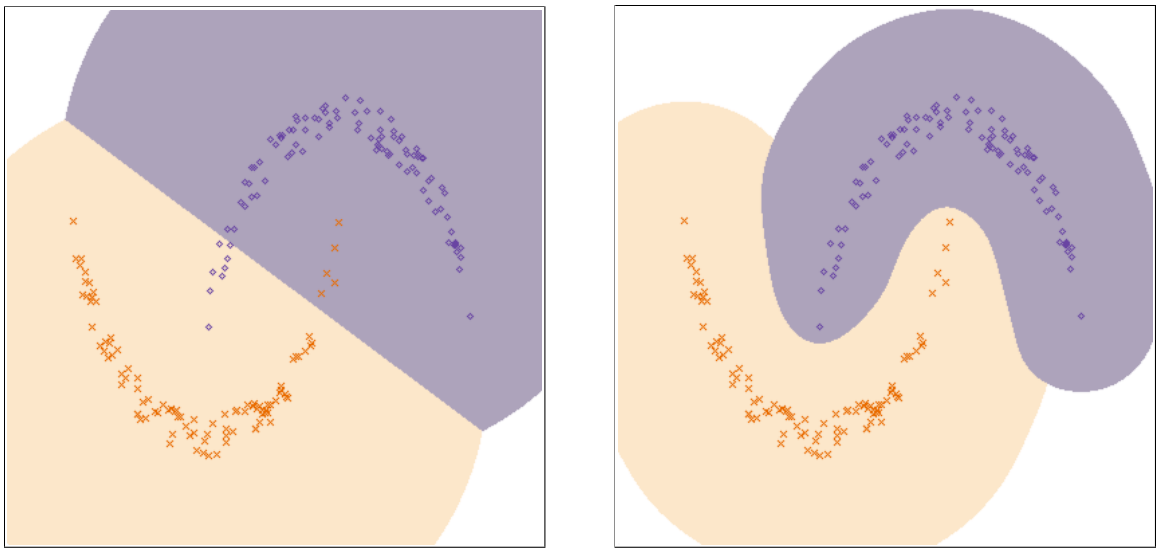
Some of the common clustering methods that can be modified for kernel clustering are the k-means algorithm, the fuzzy c-means algorithm, and the hierarchical clustering algorithm with its variants [47, 66].

- The *k*-means algorithm iteratively searches for the *k* clusters that minimize the squared distance of each point to the center of its respective cluster [47]. Each point is given membership in a single cluster in this algorithm [39]. It should be noted that this algorithm is sensitive to the chosen initial partition and may converge to a local minimum [39]. Equation (2.2) is used to calculate cluster membership in the kernel *k*-means algorithm [21, 47].

- The fuzzy c-means algorithm is similar to the k-means algorithm, except each point is allowed to have partial membership in multiple clusters and the function being minimized reflects this fact [47].

- The hierarchical clustering algorithm initializes with each point belonging to its own cluster, then iteratively joins the closest clusters together [66]. Changing the measure used to determine how close one cluster is to another has led to several variants of the hierarchical clustering algorithm. The iteration stops when all points belong to a single cluster or when k clusters have been formed.
  - The average-link algorithm joins clusters together based on the average pairwise distance between points in the clusters [47, 66].
  - The single-link algorithm joins clusters together based on the minimum pairwise distance between points in the clusters [39, 66].
  - The complete-link algorithm joins clusters together based on the maximum pairwise distance between points in the clusters [39, 66].
  - The centroid-link algorithm joins clusters together based on the distance between cluster centroids [66].

Figure 2.8 presents a visual demonstration of the differences between traditional linear clustering and kernel clustering. These figures are real-valued spaces on $[-1, 1]^2$. The background colors of these figures represent what cluster a sample in that space would be closest to using Equation (2.2). These background colors also represent which cluster a new unlabeled sample would be considered a part of. Note that for the sake of demonstration, this is an idealized case; the samples were already labeled and this represents the best case possible for any clustering algorithm. This demonstration is focusing on the differences of the Linear versus Gaussian distance measures for clustering. It is easy to see that the Gaussian distance in Figure 2.8b more accurately fits the shape of the original data.



(a) Linear distance.

(b) Gaussian distance with $\lambda = 40$.

Figure 2.8: Clustering examples over the same dataset.

2.4 Related Work

Guzella et al. [33] considered applying kernel methods to negative selection but disregarded it because evaluation of distance in feature space "merely alters the recognition region of each detector in the input space" and using a RBF kernel "can be seen as merely changing the radius of detection in comparison with the one obtained with Euclidean distance". Kernel methods do alter the recognition region of the detector, and that is the goal of this research. Figure 2.8b shows that shapes that are not hyper-spherical can be obtained from a RBF kernel if the center of the shape is not mapped directly from $\mathcal{X}$. Figure 2.8b implicitly computes the center of the hyper-sphere as the centroid in $\mathcal{H}$ of all the points belonging to the shape's cluster (according to Equation (2.2)).

Existing work combining AIS algorithms with kernel methods focus on immune network AISs. Guzella et al. [33] went on to apply kernel methods to aiNet, creating a kernel-based immune network.

In another relevant paper, Özşen et al. [61] applied kernel methods to a form of multi-class positive selection. It is unclear precisely how they implemented this solution. If Özşen et al. are using the simplest form of positive selection, the problem of only changing the radius of the detector when using a RBF kernel is not addressed. If a different algorithm is being used, then this research differs by using the NSA.

2.5 Summary

This chapter presents background topics used in this thesis including ML and the AIS. The specific AIS algorithm this research modifies is considered. Some objections against the AIS are examined. Kernel methods, as used in this investigation's modification, are discussed. Finally, this chapter concludes by reviewing previous work related to this research.

III. Implementation

This chapter describes the implementation of the Kernel Extended Real-valued Negative Selection Algorithm (KERNSA), a novel algorithm combining the Real-valued Negative Selection Algorithm (RNSA) with kernel methods. The application of kernel methods to pre-existing Machine Learning (ML) algorithms has resulted in significant improvements to the original algorithm, particularly with kernel clustering [11, 21, 47]. Because of these improvements, KERNSA explores combining kernel methods with the RNSA. This combination improves the RNSA by allowing the creation of different detector shapes in the input space, which should fit a given problem better.

3.1 Kernel Extended Real-valued Negative Selection Algorithm (KERNSA)

KERNSA, presented in Figure 3.1, differs from the Negative Selection Algorithm (NSA) (see Figure 2.5) in three key points. First, the detector's center is implicitly defined in $\mathcal{H}$ (the feature space) by using multiple points in $\mathcal{X}$ (the input space). Second, similar to the V-detector, radii are not fixed in KERNSA, but are calculated for each detector based on the nearest self sample. Finally, classifiers created by KERNSA give a continuous real-valued output instead of the NSA's discrete output. These modifications are discussed in detail in the following subsections.

3.1.1 Supporting Vectors.

Many RNSA implementations use hyper-spherical detectors [77]. KERNSA continues to use hyper-spherical detectors, but does so in $\mathcal{H}$ instead of $\mathcal{X}$. The result is that the detector can be nearly any arbitrary shape in $\mathcal{X}$, depending on the kernel used and the center of the hypersphere in $\mathcal{H}$. However, when using Radial Basis Function (RBF) kernels, if the center of the detector is mapped from a single point in $\mathcal{X}$, the kernel effectively changes the radius of the hypersphere and the detector's shape continues to be hyper-spherical in $\mathcal{X}$.

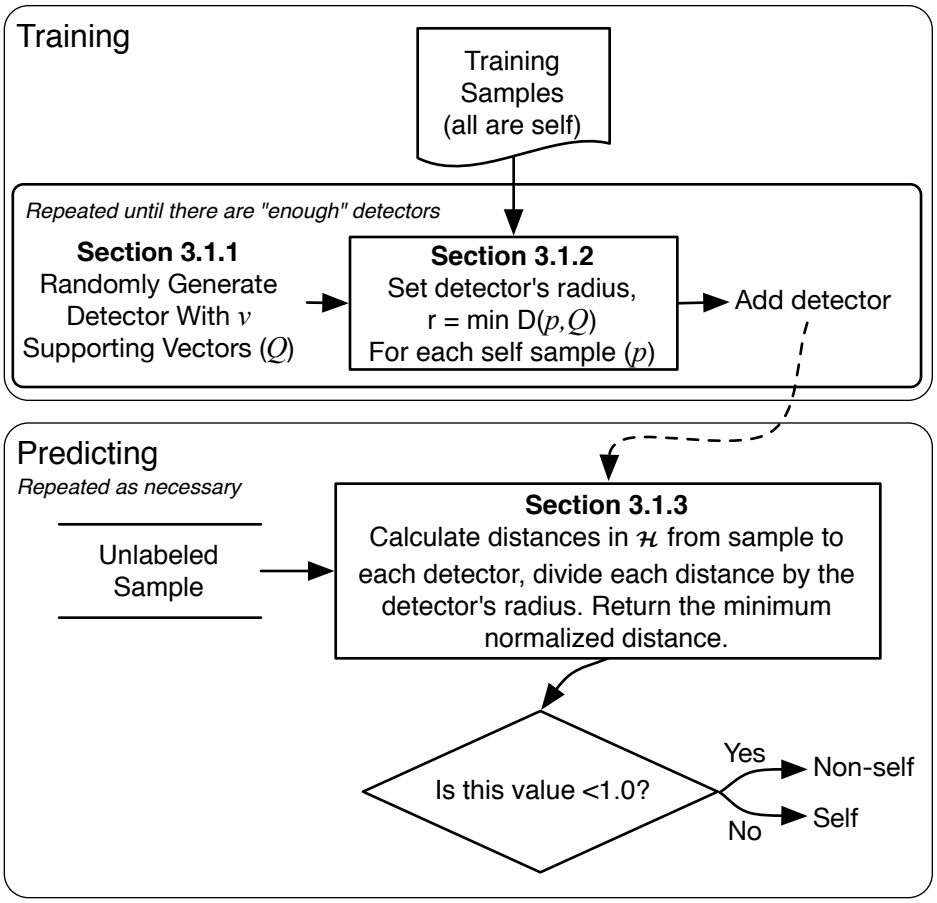


Figure 3.1: The KERNSA.

In this research, it was observed that if the center of a detector is some other arbitrary point in $\mathcal{H}$ that isn't directly mapped from $\mathcal{X}$, it is unlikely to be hyper-spherical in $\mathcal{X}$, even for RBF kernels. Thus, by finding an alternative center-point definition, the shape may be changed for RBF kernels. This property is shown in Figure 2.8b. Even though this figure is using the Gaussian RBF kernel, it forms shapes that are not hyper-spherical in $\mathcal{X}$. The centers of these shapes are implicitly mapped to a location in $\mathcal{H}$ by taking the centroid of several points in $\mathcal{X}$ according to Equation (3.1). Note that this new implicit point is unlikely to be reached from a single point in $\mathcal{X}$ mapped to $\mathcal{H}$.

$$D_k(p, Q) = \sqrt{k(p, p) + \frac{1}{n_Q^2} \sum_{q \in Q} \sum_{q' \in Q} k(q, q') - \frac{2}{n_Q} \sum_{q \in Q} k(p, q)}. \tag{3.1}$$

To bring the improvements made in kernel clustering to the RNSA, Equation (3.1) serves as the starting point. The shapes generated by Equation (3.1) seen in Figure 2.8b become "detection" areas for new samples. That is, in kernel clustering new samples that are covered by a shape are detected as belonging to that shape's associated cluster. In KERNSA, if a sample is covered by one of these detection areas then it is classified as non-self.

To gain an understanding of Equation (3.1), it is helpful to demonstrate the effect it has when using a Linear kernel. Recall that the Linear kernel is an identity function and no space transformation is performed ($\mathcal{H} = \mathcal{X}$). In this example, the set of points Q of interest consists of three two-dimensional points, [1.0, 13.0], [3.0, 12.0], and [2.0, 11.0]. The point p is located at [5.0, 12.0]. To find the distance between the centroid of Q and the point p, Equation (3.1) is used. Though Equation (3.1) implicitly calculates the centroid of Q, it can be found explicitly for ease of demonstration by averaging the points in Q together (this explicit calculation only works for the Linear kernel). The centroid of Q is thus calculated to be [2.0, 12.0]. So, the distance from the centroid of Q (which is [2.0, 12.0]) to the point p (which is [5.0, 12.0]) is 3.0 units. This example is presented in Figure 3.2.

Even though using the Linear kernel with Equation (3.1) serves as a good example, using the equation with the Linear kernel offers no detection improvement over RNSA because it still results in a hyper-spherical detector, the detector is just re-centered in $\mathcal{X}$ and favors the middle of the input space. The Linear kernel detector favors the middle of the input space because the supporting vectors are generated in $\mathcal{X}$ according to a uniformly random distribution and Equation (3.1) uses the centroid of these vectors. Imagine repeatedly rolling a six-sided die five times and taking the average; these averages will tend to be around three or four. Extending this center-favoring to the Linear kernel KERNSA is certainly not a good way for detectors to cover the available input space.

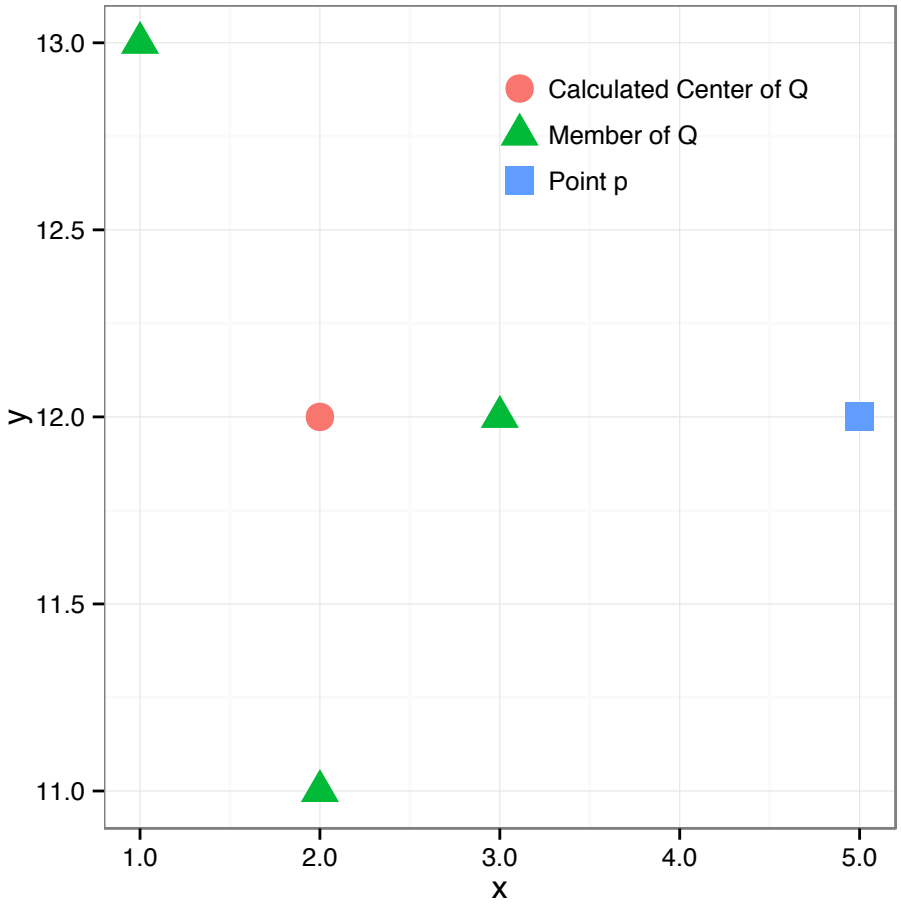


Figure 3.2: Linear kernel distance example, note the distance of 3.0 units between the center of Q and point p.

This research uses Equation (3.1) in the distance calculations in order to create shapes other than hyperspheres in $\mathcal{X}$. Since using the Linear kernel with Equation (3.1) results in center-favoring hyper-spherical detectors, this research is motivated to make an exception for the Linear kernel so that it can be given a more fair comparison to the other kernels on the basis of complexity when using more than a single supporting vector. For the Linear kernel in KERNSA, each supporting vector is treated as if it is the center of an independent hyper-spherical detector. However, a single radius is shared between all supporting vectors

of a detector. This radius sharing is done to provide a more direct comparison between the different kernels. The final equation is shown in Equation (3.2).

Let Q be a set of supporting vectors in $\mathcal{X}$. Let n_Q be the number of points in the set Q. The distance in $\mathcal{H}$ between p and the centroid in $\mathcal{H}$ of Q is

$$D_k(p, Q) = \begin{cases} \min_{q \in Q} \|p - q\| & \text{if using the Linear kernel.} \\ \sqrt{k(p,p) + \frac{1}{n_Q^2} \sum_{q \in Q} \sum_{q' \in Q} k(q, q') - \frac{2}{n_Q} \sum_{q \in Q} k(p, q)} & \text{otherwise.} \end{cases} \tag{3.2}$$

One possible Linear kernel classifier in a two dimensional space is presented in Figure 3.3. This example contains two detectors and each detector has three supporting vectors. From this, it can be calculated that there must be a total of $2 \times 3 = 6$ circles in the figure.

3.1.2 Detector Generation.

The RNSA generates fixed-radius hyperspheres at random locations in $\mathcal{X}$. If a randomly generated detector covers any self samples the detector is rejected and regenerated repeatedly until it no longer covers any self samples. The radius parameter is determined in advanced [44]; some variations include a search for the best radius [6].

KERNSA takes a different approach for the detector generation. A fixed number of supporting vectors are placed randomly in $\mathcal{X}$ and the radius of the detector is set to the distance calculated by Equation (3.2) from this new detector to the nearest self sample. Detectors are never rejected or regenerated as in the RNSA. This matches the generation of the variable sized detectors in the V-detector algorithm [42].

One concern with this method of detector generation is that very small detectors (compared to the available input space) may be generated and accepted. The V-detector algorithm compensates for this by moving detectors away from nearby self samples [42], but since this approach is not as simple with supporting vectors and since this research is exploratory, this is left as a possible improvement for future work.

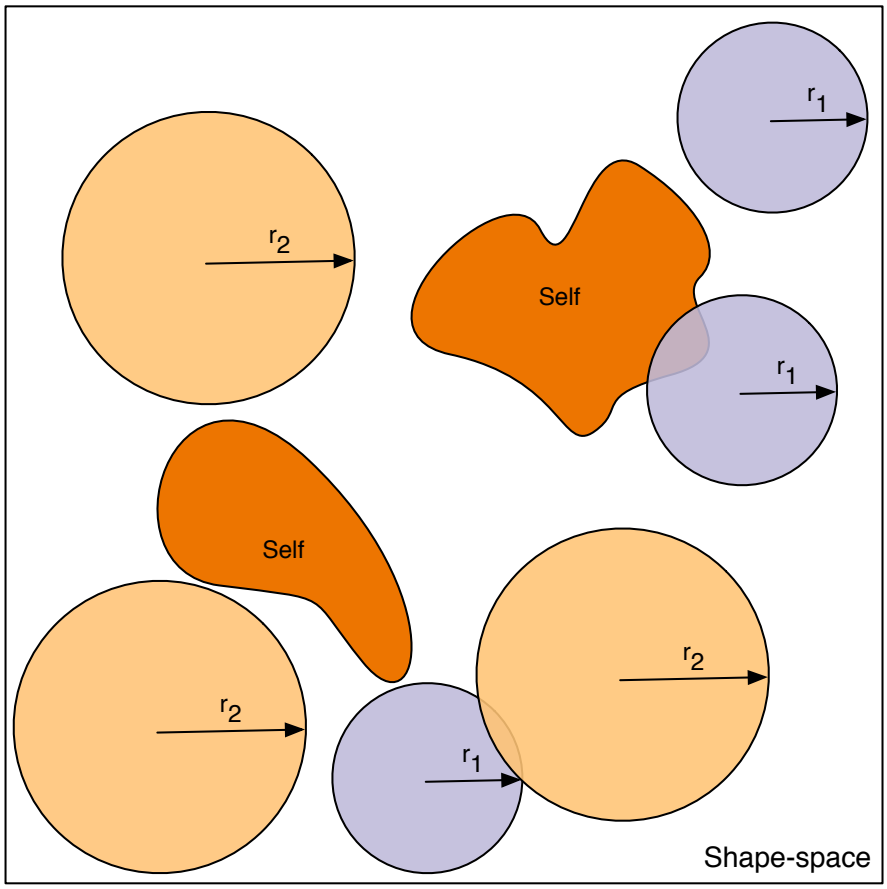


Figure 3.3: Illustration of two detectors each with three supporting vectors using the Linear kernel. Note that there are two detectors and thus only two unique radii.

3.1.3 Continuous Output.

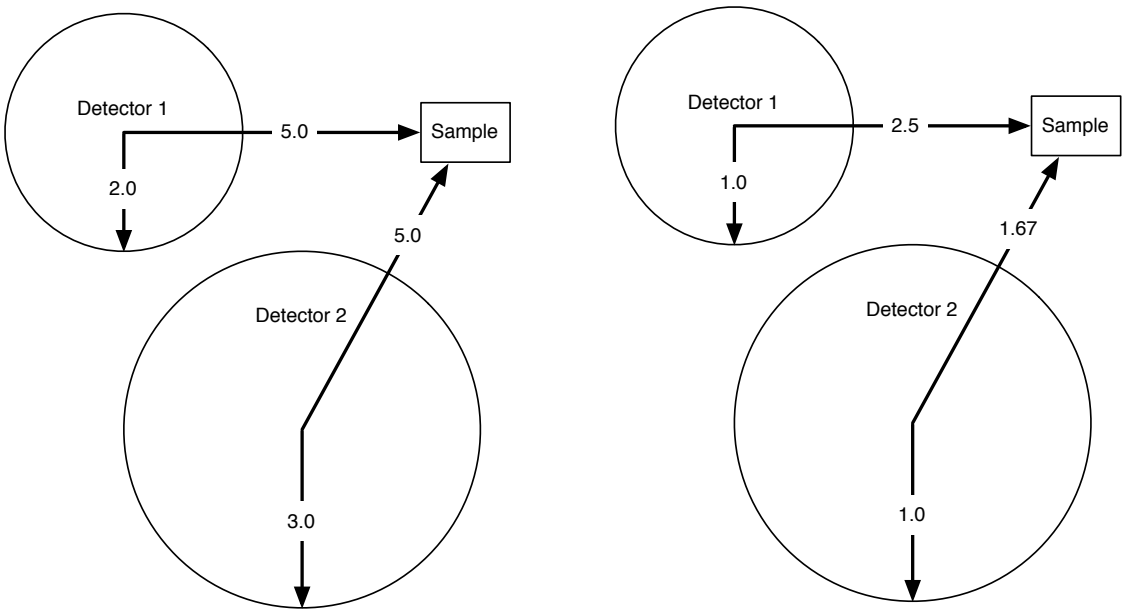
The RNSA outputs discrete predictions; either a detector covers an unknown sample and it is therefore probably non-self, or it is not covered and therefore probably self. For the RNSA, asking if any detector covers a sample is equivalent to asking if the distance between the sample and any detector is less than the radius of the detector. For example, there are two detectors $d1$ and $d2$ with $d1$'s radius = 3.0 and $d2$'s radius = 5.0. If a new sample needs to be classified and the distances to $d1$ and $d2$ are calculated to be 7.2 and 4.5 respectively, $d2$ is clearly covering this new sample and the sample should be classified as non-self. If

another new sample's distances to $d1$ and $d2$ are calculated to be 3.8 and 8.1 respectively, no available detector is covering this sample and the sample should be classified as self.

Since a continuous output measure is necessary to build a Receiver Operating Characteristic (ROC) for the experimental results, this idea of using distances was pursued. The first problem encountered is that since each detector can have a different radius, each distance value must be associated with the radius of the detector used to get the distance value. Instead of keeping this additional piece of information, KERNSA "normalizes" each distance by dividing it by the associated detector's radius. Now when given a collection of normalized distances to an unknown sample from all detectors, that sample is covered by a detector if any normalized distance is less than 1.0 (the division rescales all the distances such that the radius of each associated detector is now one unit). Likewise, if all the normalized distances are greater than or equal to 1.0, the sample in question must be self (it does not fall inside any detectors). This distance normalization is demonstrated in Figure 3.4.

The final problem for KERNSA's ROC generation is that a single output value is needed, and the discussion up to this point uses a set of normalized distances. Since the decision is made on whether *any* normalized distance is less than 1.0, KERNSA outputs the minimum value from the set of normalized distances. Thus the original RNSA decision boundary is maintained in KERNSA, but the output is now continuous to allow for ROC curve generation.

Another benefit of modifying the RNSA to have a continuous output is that it serves as a confidence level or probability in the classification. For example, a sample with a lower output is more likely to be non-self than a sample with a higher output, even if the higher output still falls under the 1.0 detection boundary. This gives a decision maker the ability to balance the True Positive Rate (TPR) versus the False Positive Rate (FPR) to the needs of a given problem domain (e.g., the cost of false negatives is generally greater than the cost of false positives when diagnosing illnesses).



(a) Before normalization.

(b) After normalization.

Figure 3.4: Distance normalization. Note that distances are normalized per detector. These normalized distances are used to build the ROC curve.

3.2 Experiment Setup

The kernels chosen for this research are the Linear, Gaussian, Inverse Multiquadratic, and Sigmoid kernels. The formulas for these kernels are repeated in Table 3.1. These particular kernels are chosen because they are relatively popular in ML literature [59, 71]. Recall that the Gaussian and Inverse Multiquadratic kernels are RBF kernels, which have special properties of interest to KERNSA. The Linear kernel is included to give a comparison to the performance of a standard RNSA. The results from the Linear KERNSA should be similar to the RNSA.

Similar to other ML algorithms, there are parameters that are varied in KERNSA to create an optimal performing classifier for a given dataset. The number of parameters to be varied is dependent on the kernel function (the affinity function) being used. All affinities

Table 3.1: The kernels chosen for this research (repeated).

Kernel Name	**Definition**	**Notes**
Linear	$\langle x, x' \rangle$	With this kernel, $\mathcal{H} = \mathcal{X}$. That is, no transformation is applied.
Gaussian [71]	$e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$	$\sigma \in \mathbb{R}$. Sometimes σ is instead represented as λ such that $\lambda = \frac{1}{2\sigma^2}$ and $\lambda \in \mathbb{R}^+$. This is a RBF kernel.
Inverse Multiquadratic [33]	$\frac{1}{\sqrt{\|x-x'\|^2 + \sigma^2}}$	$\sigma \in \mathbb{R}$. This is also a RBF kernel.
Sigmoid [51]	$\tanh(a\langle x, x' \rangle + c)$	$a, c \in \mathbb{R}$.

have at least two parameters, the number of detectors and the number of supporting vectors used. A list of parameters for each affinity function is shown in Table 3.2.

Table 3.2: Parameters for each affinity function.

Name	**Parameters**
Linear	Detectors (d) Supporting Vectors (v)
Gaussian	Detectors (d) Supporting Vectors (v) λ
Inverse Multiquadratic	Detectors (d) Supporting Vectors (v) σ
Sigmoid	Detectors (d) Supporting Vectors (v) a c

A single experiment takes as input the dataset being used, the number of detectors, the number of supporting vectors, the affinity function, and any additional parameters necessary for the affinity function. The dataset used as input must be given as k-folds or as dedicated training and testing sample sets. The University of California, Irvine (UCI) datasets discussed in Section 3.3.1 are separated into five stratified folds since they do not

have enough samples to dedicate a test set. The Dasgupta datasets discussed in Section 3.3.2 are already separated into a training and testing set.

Recall that the selection of training data and the selection of testing data are two sources of variation in estimating a classifier's performance [22]. To remove this source of variation, the UCI datasets are separated into the same five folds for each experiment regardless of the parameters or affinity function used.

A single experiment outputs the resulting classifier, the estimated Area Under Curve (AUC) for the ROC, and the estimated accuracy for each label. Each label in the dataset has a separate classifier created with that label treated as self, thus there is a distinct set of results for each label. The final classifier that is output from the learner is what would be used to classify future samples if one was using the KERNSA in an active system. The Dasgupta datasets only have one label and thus only one set of outputs. Figure 3.5 presents this experiment process as discussed.

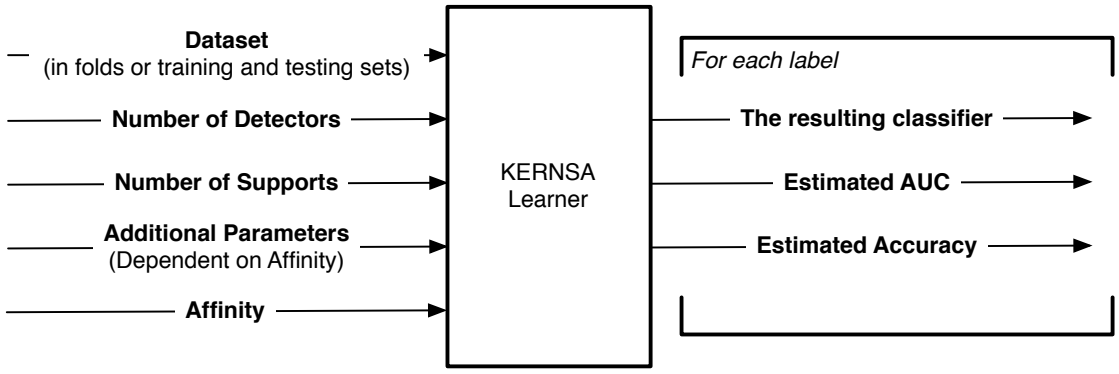


Figure 3.5: A single KERNSA experiment.

3.2.1 Parameter Sweep.

The parameters used in KERNSA need to be tuned to ensure optimal results for a given dataset. These parameters are tuned by starting with a base set of parameters and running one experiment on each possible combination of parameters. This base set is chosen

by taking values near the defaults used in most ML algorithms. The results are examined and additional parameter values are added to the experiment if a pattern of increasing performance is observed near the previously unexplored parameter values. The process is repeated as necessary, adding additional parameter values at each iteration.

The process of examining and adding additional parameter values is manual. After running an experiment for each combination of parameter values, a value grid of AUCs is generated in order to visualize patterns in the results and identify values that should be added. An example of this value grid can be seen in Figure 3.6. This particular grid belongs to the results from the Iris dataset using the Inverse Multiquadratic affinity function. One can see in this figure that the highest AUC values have been found and the values decrease near the edges of the parameter space.

3.2.2 Accuracy Calculation.

Though this research uses the AUC as the primary means of evaluating and comparing learning algorithms, accuracy is also calculated in order to offer comparisons to previous research. This research calculates accuracy one of two ways depending on the number of labels in the dataset.

The first method of accuracy calculation is used when there are only one or two labels. The KERNSA is trained on one label as self and all other labels are considered non-self during testing. If any detector covers a tested sample, that sample is predicted to be non-self and self otherwise. This method of calculation more closely follows the original intent of the NSA and is therefore preferred. In the case of two labels, the learning algorithm is run on each label such that first run treats the first label as self and the second run treats the second label as self. The overall best performing classifier (by AUC) after tuning is used to calculate the accuracy for that dataset. Accuracy for these datasets is defined as

$$\text{Accuracy} = \max_{l \in L} \frac{\text{Number of correct predictions when using } l \text{ as self}}{\text{Total number of samples}},$$

where L is the set of unique labels used in a dataset.

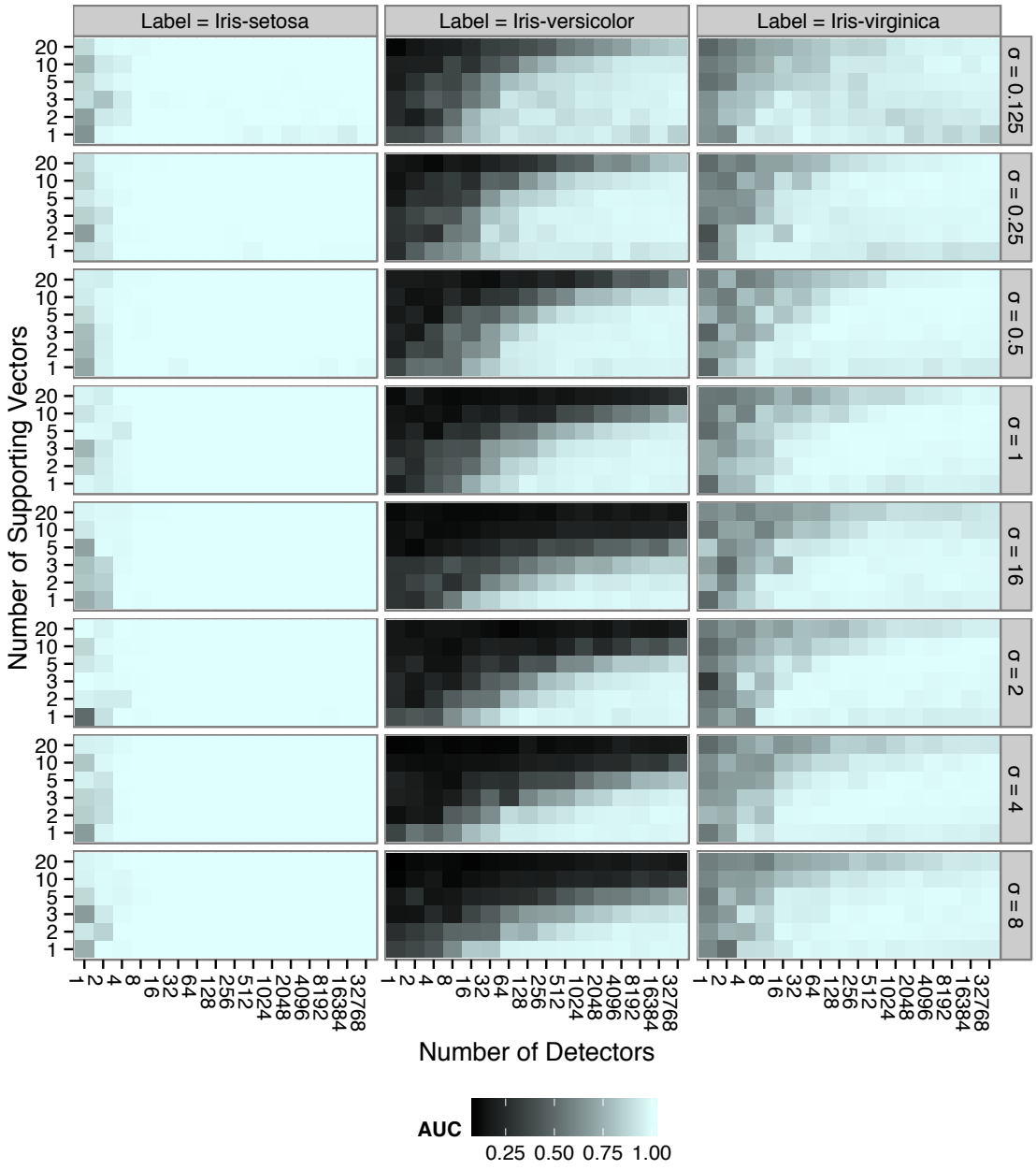


Figure 3.6: Example of value grid used in parameter sweeps.

The second method of accuracy calculation is used when there are three or more labels in the dataset. The NSA was not created with these kinds of problems in mind, but it is still possible to use it to make predictions. For each label, the best performing classifier after

tuning is chosen, creating the set of classifiers C. When predicting a label, each classifier's output is taken into consideration. The sample is labeled as the label belonging to whichever classifier has the largest output. Keep in mind that each classifier is trained with samples belonging to a single label and that the output from a classifier is the normalized distance from the nearest detector. Thus, whichever classifier gives the largest output is indicating that a large margin of space exists between that classifier's detectors (which determine non-self) and this sample. The accuracy for these datasets is calculated as,

$$\text{IsCorrect}(\textit{sample}) = \begin{cases} 1 & \text{if } \max_{c \in C} \left[\text{PredictionFor}(c, \text{sample})\right] \text{ comes} \\ & \text{from the classifier associated with sample's label} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Accuracy} = \frac{\sum_{s \in \text{samples}} \text{IsCorrect}(s)}{\text{Total number of samples}}.$$

This second method is demonstrated in Figure 3.7. In this figure, the sample would be classified as C, since the detector associated with that label is the furthest away. If the actual label of the sample is C, this would be considered a correct classification for the accuracy calculation.

3.2.3 Determining The Top Performing Classifier.

After completing the parameter sweep, the ten best (by AUC) parameter settings for each affinity function and label are chosen. Thirty additional experiment repetitions are performed on these top performers to better estimate their AUCs and accuracies. These repetitions are averaged together and the overall top performer by AUC is taken as the "best" tuning of the parameters for the dataset. Though not recorded in this investigation, the outputs from the individual repetitions are kept for the analysis of the results. The top performer is what is used in the analysis in the following chapter.

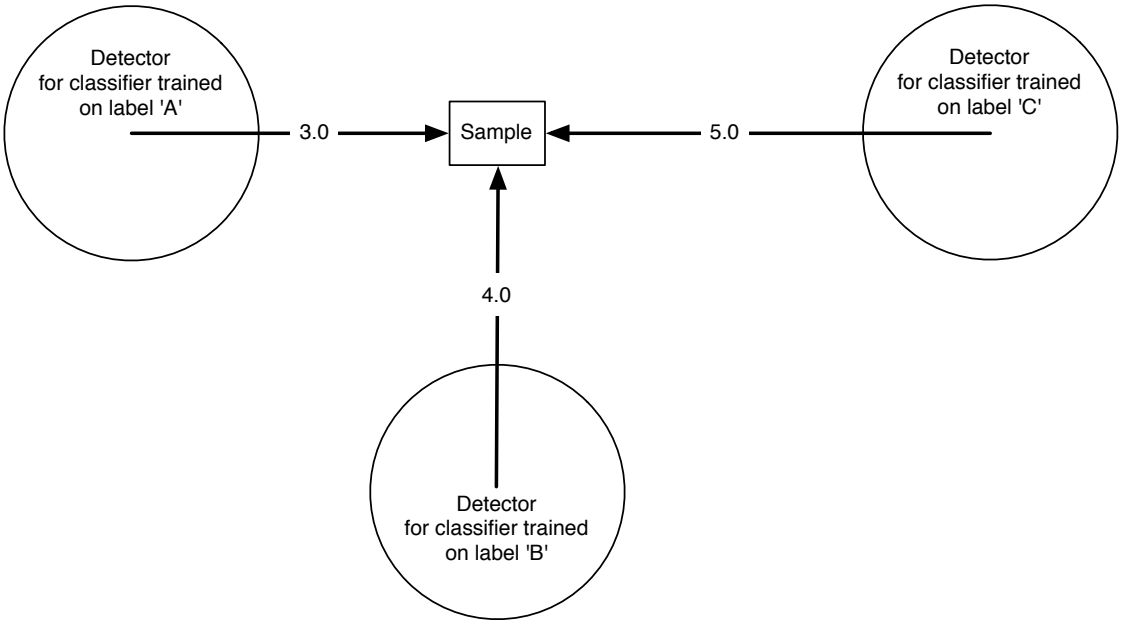


Figure 3.7: Sample classification to calculate accuracy when a dataset has more than two unique labels.

3.3 Datasets

When comparing classifiers, one expects that no single classifier performs better than others on all possible datasets. Bias-free learning is futile and there is "no free lunch" in ML [84]. Theoretically, no learning algorithm can outperform all others on all possible problem domains [40]. In fact, even a poorly performing algorithm can perform significantly better than other algorithms if an appropriate dataset is created [50]. It is impossible to demonstrate an algorithm's superiority in all cases, so the strengths of one algorithm versus another should be identified instead [40]. Since this research is identifying the performance of KERNSA versus RNSA in general, care must be taken to choose datasets that are representative of the datasets that would be used in RNSA applications. Toward this end, this research uses a combination of repository-based and artificial datasets.

3.3.1 Repository-based Datasets.

A dataset repository gathers datasets in a single location so that researchers may access these datasets in the future for replication or comparison of results. Some of the advantages of using repository-based datasets include:

- The datasets found in repositories are generally real-world data [40].
- Researchers can immediately use these datasets without expert knowledge of the domain [40].
- Since these datasets are public, comparisons between algorithms and replications of previous experiments can be made more easily [40].
- These repositories are constantly growing and reflect new concerns for ML [40].

There have also been some concerns raised about the use of these repositories. These arguments include:

- Since the exact distribution over $\mathcal{X}$ and the associated sampling noise is unknown, conclusions from repository-based experiments are questionable [40].
- Even though these repositories represent problems that often arise in practice, they do not represent the whole range of expected real-world problems [36].
- When making comparisons between learning algorithms, familiarity with an algorithm can give an unfair advantage to that algorithm [40]. Since the researcher may know how to tune a specific algorithm to get the best performance but may not have the knowledge required to tune the other algorithms they are comparing to, it is natural that an advantage would arise for the properly tuned algorithm. Note that this is not a direct attack against the use of the repositories, it is only pointing out the care that should be taken in comparing algorithms.

- Since these datasets are intended for public use, they are simplified and of limited complexity with difficult cases sometimes removed [40].

Given these arguments, this research uses the UCI ML repository [1]. This repository has had a major positive impact on the way ML research is conducted by improving the quality and thoroughness of the research [2]. The UCI repository tends to absorb all other datasets from other repositories and is generally considered the authoritative repository of ML research [40].

The chosen datasets from the UCI are the diabetes, iris, ionosphere, sonar, Wisconsin Diagnostic Breast Cancer (WDBC), and wine datasets. These datasets are chosen because they are used in other Artificial Immune System (AIS) research [9, 10, 24, 55] and thus KERNSA's performance can be compared with them. Table 3.3 presents some important statistics on these datasets.

Table 3.3: Sizes associated with chosen repository-based datasets.

Name	**Number of Samples**	**Number of Features**	**Number of Labels**
Diabetes	768	8	2
Iris	150	4	3
Ionosphere	351	34	2
Sonar	208	60	2
WDBC	569	30	2
Wine	178	13	3

- The **diabetes** dataset has features representing various measures from a human patient (number of times pregnant, skin thickness, age, etc). Each sample's label indicates if the patient has diabetes.

- The **iris** dataset is commonly used in ML literature. The features are various measurements of an iris plant. Each sample's label is the specific type of iris the plant belongs to. The three types of iris plants under investigation are iris-setosa, iris-versicolor, and iris-virginica. There are fifty instances of each label. Iris-setosa is linearly separable from the others.

- The **ionosphere** dataset is taken from radar bounces in the ionosphere. The label is whether the radar signal is returned to the sensor or passes through the atmosphere.

- The **sonar** dataset has features that are representative of the signal returned from a sonar chirp. These chirps are created from various angles around either a cylinder or a rock. The label is whether the object is a cylinder, which approximates a land mine, or a rock.

- The **WDBC** dataset is a set of cancer diagnoses. Each sample in this dataset is taken from a patient's breast mass. The features are various physical measurements of this mass. Each sample's label is whether the mass is malignant or benign.

- In the **wine** dataset, the features are the chemical analysis of samples of wine. The label is the origin of the wine.

3.3.2 Artificial Datasets.

By using an artificial dataset, the distribution of the samples from X and the associated noise are set a priori, which addresses the first argument against using repository-based datasets. Since artificial datasets can be tweaked, introducing a single change at a time, using artificial datasets allows ML literature to better explore what types of datasets might be appropriate problems for a given learning algorithm.

Dasgupta [15, 44] has created several two-dimensional artificial datasets aimed specifically at one-class classifiers. These datasets have been used in other literature [6, 30,

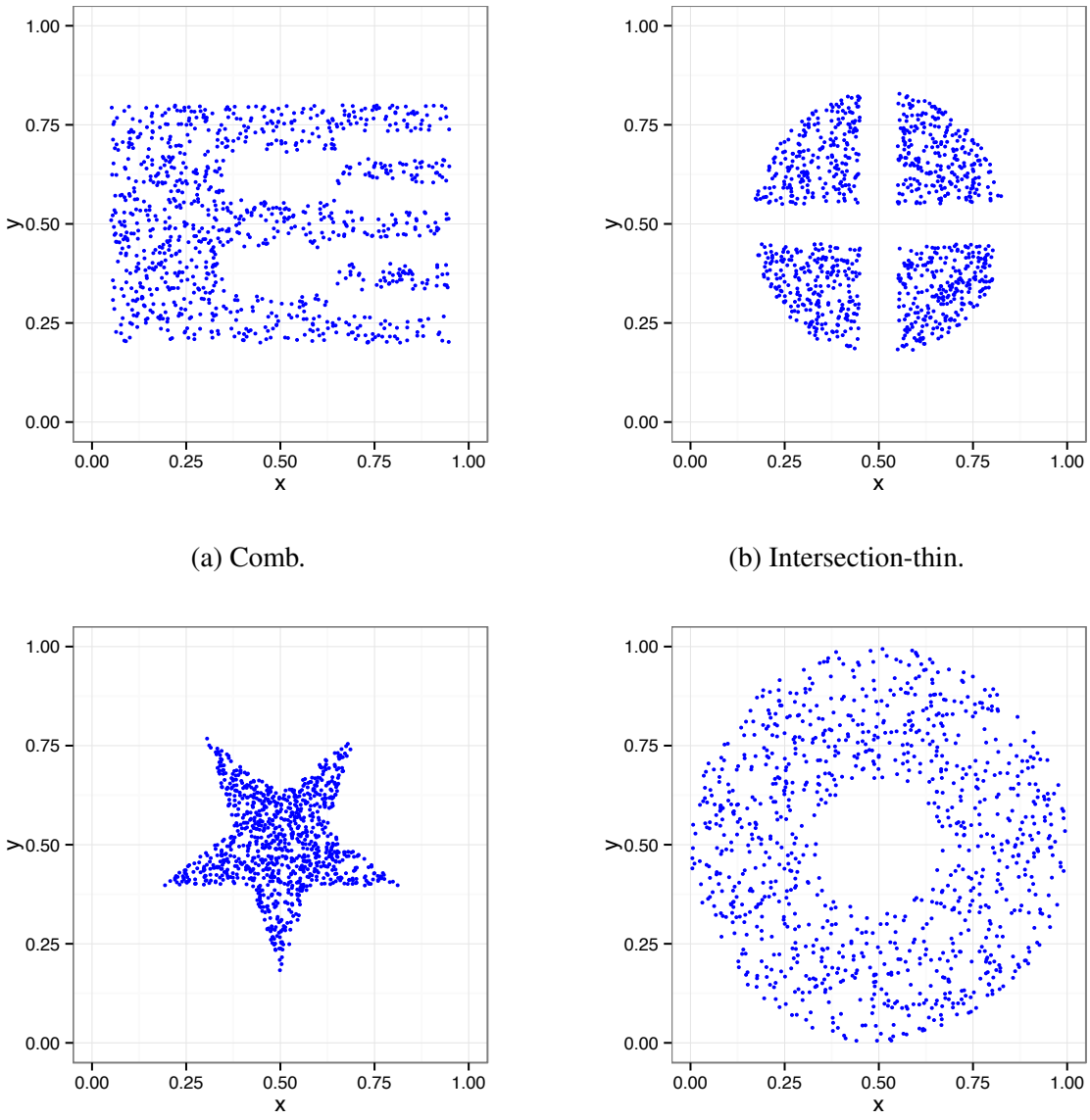
43, 44, 85], and are publicly accessible. The datasets from Dasgupta [15, 44] chosen for this research are the comb (Figure 3.8a), intersection-thin (Figure 3.8b), pentagram-mid (Figure 3.8c), and ring-thick datasets (Figure 3.8d).

Each of the Dasgupta datasets has 1,000 samples of self that are used for training the classifier. Each dataset also has 1,000 samples used for testing which include samples of both self and non-self. The self data for training from each chosen dataset can be seen in Figure 3.8.

3.3.3 Dataset Preprocessing.

Each dataset is normalized so that all samples fall inside the hypercube $[-1, 1]^n$. Note that normalization may introduce distortions or biases into the dataset [65]. However, KERNSA relies heavily on distance information and normalization ensures that distance in one feature is as meaningful in another. Otherwise, the dataset may have one feature in the range $[0.0, 1.0]$ and another in the range $[0.0, 10000.0]$. Without normalization, distance on the second feature would likely be considered more important by KERNSA. In addition, the Sigmoid kernel (which uses the hyperbolic tangent function) runs into precision problems with larger numbers. Finally, feature scaling has been shown to improve classifier performance [80].

The UCI datasets contain features that are intended to uniquely identify a sample. Since the connection of these identifiers to a sample is only artificial (they are not sampled from the problem space), these features are dropped before being processed by KERNSA.



(a) Comb.

(b) Intersection-thin.

(c) Pentagram-mid.

(d) Ring-thick.

Figure 3.8: Chosen Dasgupta datasets

3.4 Summary

This chapter discussed the modifications made to the NSA to build the KERNSA. The three changes made are the inclusion of supporting vectors, the variable length radii in detector generation, and the continuous output from the classifier (see Figure 3.1). The

experiment setup for this research was discussed, which includes the way parameters are tuned for the learning algorithm, the method used to calculate a classifier's accuracy, and how the top performing classifier is determined. This chapter concludes by discussing the datasets chosen for the experiments and how they are preprocessed.

IV. Results and Analysis

This chapter presents the results of the Kernel Extended Real-valued Negative Selection Algorithm (KERNSA) experiments outlined in Chapter 3. The optimal parameters as found in the parameter sweep are presented. The top-performing classifiers per affinity function from each dataset and each label (hereafter referred to as the dataset-label pair) are compared and statistical significance is assessed. Finally, the accuracies from the University of California, Irvine (UCI) datasets are compared to other Artificial Immune System (AIS) learning algorithms.

4.1 Parameter Sweep

The parameters used in the parameter sweep for each kernel and dataset in the experiments are listed in Tables 4.1 to 4.4. Recall that every possible combination of these parameters are tested by KERNSA while looking for the set of parameters with the best performance, as discussed in Section 3.2.

During the parameter tuning it was discovered that performance maxed out at around 30,000 to 100,000 detectors for these datasets. A maximum of twenty supporting vectors seems to offer the best performance. Additional supporting vectors may be of greater utility if they are placed in some logical pattern in response to the self data, instead of the current random placement.

Table 4.1: Number of supporting vectors and detectors searched for parameter tuning.

	Detectors (d)	Supporting Vectors (v)
Iris	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768	1, 2, 3, 5, 10, 20
Ionosphere	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072	1, 3, 5, 10, 20
Diabetes	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	1, 3, 5, 10, 20
Sonar	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072	1, 3, 5, 10, 20
WDBC	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768	1, 3, 5, 10, 20, 30
Wine	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072	1, 3, 5, 10, 20, 30
Comb	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	1, 3, 5, 10, 20
Intersection	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768	1, 3, 5, 10, 20
Pentagram	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	1, 3, 5, 10, 20
Ring	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768	1, 3, 5, 10, 20

Table 4.2: The parameters used for the Gaussian kernel.

	λ
Iris	0.296, 0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063, 7.593
Ionosphere	0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063
Diabetes	0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063
Sonar	0.198, 0.296, 0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063
WDBC	0.039, 0.059, 0.088, 0.132, 0.198, 0.296, 0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063, 7.593
Wine	0.198, 0.296, 0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063
Comb	0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063, 7.593, 11.39, 17.094, 25.641, 38.462, 57.803
Intersection	0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063, 7.593, 11.39, 17.094, 25.641, 38.462, 57.803
Pentagram	0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063, 7.593, 11.39, 17.094, 25.641, 38.462, 57.803
Ring	0.444, 0.667, 1, 1.5, 2.25, 3.375, 5.063, 7.593, 11.39, 17.094, 25.641, 38.462, 57.803

Table 4.3: The parameters used for the Inverse Multiquadratic kernel.

	σ
Iris	0.125, 0.25, 0.5, 1, 2, 4, 8, 16
Ionosphere	0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2
Diabetes	0.0625, 0.125, 0.25, 0.5, 1
Sonar	0.0625, 0.125, 0.25, 0.5, 1
WDBC	0.0312, 0.0625, 0.125, 0.25, 0.5, 1
Wine	0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32
Comb	0.0078, 0.0156, 0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2
Intersection	0.0078, 0.0156, 0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2
Pentagram	0.0078, 0.0156, 0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2
Ring	0.0078, 0.0156, 0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4

Table 4.4: The parameters used for the Sigmoid kernel.

	a	*c*
Iris	0.125, 0.25, 0.5	-2, -1, -0.5, 0, 0.5, 1, 2, 4, 8, 16
Ionosphere	0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4	0.25, 0.5, 1, 2
Diabetes	0.125, 0.25, 0.5, 1, 2	-2, -1, -0.5, 0, 0.5, 1, 2
Sonar	0.0156, 0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2, 4	-16, -8, -4, -2, -1, -0.5, 0, 0.5, 1, 2
WDBC	0.0156, 0.0312, 0.0625, 0.125, 0.25, 0.5, 1, 2	-2, -1, -0.5, 0, 0.25, 0.5, 1, 2, 4, 8
Wine	0.0625, 0.125, 0.25, 0.5, 1, 2	-2, -1, -0.5, 0, 0.5, 1, 2
Comb	0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32	-16, -8, -4, -2, -1, -0.5, 0, 0.5, 1, 2, 4, 8, 16
Intersection	0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32	-16, -8, -4, -2, -1, -0.5, 0, 0.5, 1, 2, 4, 8, 16
Pentagram	0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32	-16, -8, -4, -2, -1, -0.5, 0, 0.5, 1, 2, 4, 8, 16
Ring	0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32	-32, -16, -8, -4, -2, -1, -0.5, 0, 0.5, 1, 2, 4, 8, 16, 32

4.2 Comparing the Affinity Performances

Recall that the UCI datasets contain more than one label for every dataset and each label is trained individually, giving one set of results for every label. Tables 4.5 to 4.22 present the top performing classifier per affinity function for each dataset-label pair with the classifier's number of detectors (d), number of supporting vectors (v), and additional parameters listed depending on the affinity function. The overall top performing affinity function for each dataset-label pair is bolded.

Table 4.5: Best parameters for dataset Iris-setosa.

Affinity	Params	d	v	Area Under Curve (AUC)	95% CI
Linear	N/A	32	2	0.999857	0.999737-0.999953
Gaussian	$\lambda = 0.667$	16	1	0.999347	0.998553-0.999897
I. Multiq.	$\sigma = 4$	16	1	**0.999920**	0.999820-0.999990
Sigmoid	$a = 0.125$, $c = 0.5$	8	1	0.998897	0.998087-0.999537

Table 4.6: Best parameters for dataset Iris-versicolor.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	16384	1	0.977633	0.975447-0.979727
Gaussian	$\lambda = 2.25$	16384	3	**0.987020**	0.985507-0.988420
I. Multiq.	$\sigma = 0.5$	16384	3	0.985133	0.983460-0.986713
Sigmoid	$a = 0.5$, $c = 0$	2048	1	0.982547	0.980667-0.984280

Table 4.7: Best parameters for dataset Iris-virginica.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	64	1	0.982340	0.979414-0.985063
Gaussian	$\lambda = 0.667$	16384	10	**0.995087**	0.994017-0.996083
I. Multiq.	$\sigma = 1$	16384	10	0.994703	0.993640-0.995680
Sigmoid	$a = 0.125$, $c = -0.5$	8192	3	0.986983	0.984707-0.989150

Table 4.8: Best parameters for dataset Ionosphere-b.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	8	20	**0.610021**	0.598278-0.621542
Gaussian	$\lambda = 0.444$	16384	10	0.606627	0.596198-0.617165
I. Multiq.	$\sigma = 0.0625$	1	5	0.609261	0.598376-0.620404
Sigmoid	$a = 4$, $c = 0.25$	4096	10	0.603798	0.585773-0.622082

Table 4.9: Best parameters for dataset Ionosphere-g.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	131072	1	0.781029	0.778116-0.783881
Gaussian	$\lambda = 1.5$	32768	1	0.707854	0.702554-0.713267
I. Multiq.	$\sigma = 0.0625$	131072	1	0.779304	0.776121-0.782374
Sigmoid	$a = 0.0625$, $c = 2$	16384	1	**0.985804**	0.984999-0.986606

Table 4.10: Best parameters for dataset Diabetes-0.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	16384	10	0.729266	0.725042-0.733408
Gaussian	$\lambda = 2.25$	16384	20	**0.734657**	0.730856-0.738383
I. Multiq.	$\sigma = 1$	16384	1	0.708248	0.704109-0.712407
Sigmoid	$a = 0.25$, $c = 0$	4096	1	0.718500	0.711838-0.724895

Table 4.11: Best parameters for dataset Diabetes-1.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	256	1	0.507911	0.498934-0.517254
Gaussian	$\lambda = 5.063$	256	10	0.475130	0.465682-0.484913
I. Multiq.	$\sigma = 0.25$	512	1	0.520434	0.512045-0.528909
Sigmoid	$a = 0.25$, $c = 1$	1024	10	**0.618873**	0.603443-0.633532

Table 4.12: Best parameters for dataset Sonar-M.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	131072	20	0.442080	0.434330-0.449723
Gaussian	$\lambda = 1.5$	8	1	0.501274	0.499793-0.502634
I. Multiq.	$\sigma = 1$	131072	1	0.433983	0.426220-0.441674
Sigmoid	$a = 1$, $c = -8$	131072	5	**0.823563**	0.816854-0.830352

Table 4.13: Best parameters for dataset Sonar-R.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	65536	1	0.733925	0.724604-0.743068
Gaussian	$\lambda = 0.444$	131072	10	0.717585	0.707436-0.727421
I. Multiq.	$\sigma = 0.0625$	32768	1	0.732279	0.723006-0.741307
Sigmoid	$a = 0.0312$, $c = 0$	16384	1	**0.737498**	0.727833-0.746864

Table 4.14: Best parameters for dataset WDBC-B.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	8192	1	0.979958	0.978910-0.981008
Gaussian	$\lambda = 0.059$	1024	1	0.979580	0.978515-0.980662
I. Multiq.	$\sigma = 0.0625$	1024	1	0.979673	0.978559-0.980757
Sigmoid	$a = 0.0312$, $c = 2$	8192	3	**0.980471**	0.979446-0.981490

Table 4.15: Best parameters for dataset WDBC-M.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	32768	30	0.200531	0.195530-0.205822
Gaussian	$\lambda = 7.593$	2	30	0.500004	0.500000-0.500011
I. Multiq.	$\sigma = 0.0312$	32768	1	0.189414	0.183897-0.195202
Sigmoid	$a = 1, c = 8$	4096	30	**0.945648**	0.942229-0.948977

Table 4.16: Best parameters for dataset Wine-1.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	131072	3	0.996482	0.995770-0.997144
Gaussian	$\lambda = 0.198$	131072	1	0.994391	0.993164-0.995495
I. Multiq.	$\sigma = 16$	131072	1	0.996494	0.995732-0.997188
Sigmoid	$a = 0.125, c = 0.5$	131072	1	**0.997976**	0.997506-0.998405

Table 4.17: Best parameters for dataset Wine-2.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	16384	1	0.937436	0.932435-0.942361
Gaussian	$\lambda = 0.444$	65536	3	0.933308	0.929234-0.937467
I. Multiq.	$\sigma = 16$	4096	1	0.937003	0.931951-0.942012
Sigmoid	$a = 0.5, c = 0$	65536	30	**0.961932**	0.957675-0.966031

Table 4.18: Best parameters for dataset Wine-3.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	131072	1	0.997003	0.996311-0.997612
Gaussian	$\lambda = 0.667$	131072	5	0.996920	0.996208-0.997532
I. Multiq.	$\sigma = 16$	65536	1	0.997240	0.996549-0.997866
Sigmoid	$a = 0.25, c = 1$	131072	20	**0.998283**	0.997870-0.998634

Table 4.19: Best parameters for dataset Comb.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	1024	1	0.930324	0.928674-0.931978
Gaussian	$\lambda = 38.462$	16384	10	**0.986127**	0.985935-0.986322
I. Multiq.	$\sigma = 0.125$	16384	10	0.984082	0.983816-0.984347
Sigmoid	$a = 1, c = -1$	1024	1	0.930600	0.929068-0.932061

Table 4.20: Best parameters for dataset Intersection.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	1024	1	0.982042	0.981168-0.982911
Gaussian	$\lambda = 38.462$	32768	20	**0.999181**	0.999158-0.999203
I. Multiq.	$\sigma = 0.25$	32768	10	0.999150	0.999123-0.999176
Sigmoid	$a = 1, c = -2$	16384	3	0.996710	0.996464-0.996935

Table 4.21: Best parameters for dataset Pentagram.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	256	1	0.996925	0.996487-0.997340
Gaussian	$\lambda = 7.593$	16384	10	0.999929	0.999925-0.999933
I. Multiq.	$\sigma = 0.25$	16384	10	**0.999932**	0.999928-0.999935
Sigmoid	$a = 0.5, c = -1$	8192	3	0.999552	0.999479-0.999619

Table 4.22: Best parameters for dataset Ring.

Affinity	Params	d	v	AUC	95% CI
Linear	N/A	512	1	0.926698	0.925050-0.928332
Gaussian	$\lambda = 38.462$	32768	10	0.988587	0.988238-0.988927
I. Multiq.	$\sigma = 0.125$	32768	5	0.984213	0.983874-0.984543
Sigmoid	$a = 2, c = 16$	32768	3	**0.989525**	0.989074-0.989962

4.2.1 Statistical Testing.

It is important to note that each dataset-label pair is statistically independent of all others since the samples that are used for training the classifier are unique to each dataset-label pair. If there was not independence, special consideration would need to be given to the statistical tests used and on the aggregation of the test results. However, the statistical tests that are used *cannot* assume that the AUCs are drawn from a normal distribution, which is generally an invalid assumption when comparing Machine Learning (ML) performances [20, 40]. The traditional Analysis of Variance (ANOVA) and t-tests assume normality, and therefore are not used in this research.

Table 4.23 summarizes the information given in Tables 4.5 to 4.22 by listing the AUCs by affinity function and dataset-label pair. Each AUC in this table is followed by the rank (one through four) of the affinity function compared to the other affinity functions for the given dataset-label pair. Additionally, each AUC (except for those from the Linear affinity function) is tested to see if it is statistically different from the corresponding Linear affinity function's AUC (recall that the Linear affinity function is representative of the Real-valued Negative Selection Algorithm (RNSA)). An ANOVA with resampling test (which avoids the assumption of normality) is used to detect these differences at a 95% confidence level. If a statistical difference is detected and an AUC is *greater* than the corresponding Linear affinity function AUC, the value is bolded. If a statistical difference is detected and an AUC is *less* than the corresponding Linear affinity function AUC, a † is shown. No emphasis is added when no statistical difference is detected.

Table 4.23: AUCs from best performing parameter tuning by dataset and affinity function.

	Linear	Gaussian	I. Multiq.	Sigmoid
Iris-setosa	0.99986{2}	0.99935{3}	0.99992{1}	0.99890{4}†
Iris-versicolor	0.97763{4}	**0.98702**{1}	**0.98513**{2}	**0.98255**{3}
Iris-virginica	0.98234{4}	**0.99509**{1}	**0.99470**{2}	**0.98698**{3}
Ionosphere-b	0.61002{1}	0.60663{3}	0.60926{2}	0.60380{4}
Ionosphere-g	0.78103{2}	0.70785{4}†	0.77930{3}	**0.98580**{1}
Diabetes-0	0.72927{2}	**0.73466**{1}	0.70825{4}†	0.71850{3}†
Diabetes-1	0.50791{3}	0.47513{4}†	**0.52043**{2}	**0.61887**{1}
Sonar-M	0.44208{3}	**0.50127**{2}	0.43398{4}†	**0.82356**{1}
Sonar-R	0.73393{2}	0.71758{4}†	0.73228{3}	0.73750{1}
WDBC-B	0.97996{2}	0.97958{4}	0.97967{3}	**0.98047**{1}
WDBC-M	0.20053{3}	**0.50000**{2}	0.18941{4}†	**0.94565**{1}
Wine-1	0.99648{3}	0.99439{4}†	0.99649{2}	**0.99798**{1}
Wine-2	0.93744{2}	0.93331{4}	0.93700{3}	**0.96193**{1}
Wine-3	0.99700{3}	0.99692{4}	0.99724{2}	**0.99828**{1}
Comb	0.93032{4}	**0.98613**{1}	**0.98408**{2}	0.93060{3}
Intersection	0.98204{4}	**0.99918**{1}	**0.99915**{2}	**0.99671**{3}
Pentagram	0.99692{4}	**0.99993**{2}	**0.99993**{1}	**0.99955**{3}
Ring	0.92670{4}	**0.98859**{2}	**0.98421**{3}	**0.98953**{1}
Average Rank	2.889	2.611	2.5	2

The affinity function's rank for each dataset-label pair is shown in superscript.

A bold value is statistically better than the Linear affinity function for that dataset-label.

The † indicates the value is statistically worse than to the Linear affinity function for that dataset-label.

The ranks from each affinity function across all datasets are averaged and displayed in Table 4.23. Note that the Linear affinity function takes last place with an average rank of 2.89, though it is not far behind the other affinities. Sigmoid takes the lead with an average rank of 2. Note that all but three dataset-label pairs have at least one affinity function which performs better than the Linear affinity function with statistical significance. Additionally, the Linear affinity function is never statistically better than all three of the other affinity functions.

4.3 Pattern Analysis

The Sigmoid affinity function performs surprisingly well over these datasets. Sigmoid's performance is particularly evident on the datasets where other affinities perform overwhelmingly poorly. For example, with the Wisconsin Diagnostic Breast Cancer (WDBC)-M dataset the Linear, Gaussian, and Inverse Multiquadratic kernels have AUCs of 0.2005, 0.5, and 0.1894 respectively, while Sigmoid has an AUC of 0.9456. A similar pattern of the Sigmoid's performance can be seen in Ionosphere-g, Diabetes-1, and Sonar-M. This seems to indicate that there are some problem domains that the Sigmoid KERNSA performs exceptionally well with in comparison to the other affinities.

The Linear affinity function outperforms the other affinities in only one dataset-label (and even then the differences aren't significant), but in general it performs better than expected. It is interesting that the best performing parameter tuning for the Linear affinity function usually has a single supporting vector, and some of the cases that use more than one vector (Ionosphere-b, Sonar-M, WDBC-M) may be due to KERNSA's poor performance on that dataset-label pair in general. This may indicate that the benefit from additional hypersphere detection areas with unique radii far outweighs the benefit from additional hypersphere detection areas with shared radii, at least when placing supporting vectors randomly.

Another interesting pattern seen in the results is that KERNSA performs poorly on some dataset-label pairs, but generally performs acceptably on other labels from that dataset. For example, Ionosphere-b, Diabetes-1, Sonar-M (except for Sigmoid), and WDBC-M (excepting Sigmoid again) all have AUCs at around 0.6 or less, which is not much better than guessing (recall that guessing gives an AUC of 0.5). But for these datasets, training on the other label as self gives acceptable AUCs. This pattern is even more interesting when the meaning of the labels are taken into consideration. The label "b" in Ionosphere are the "bad" instances ("g" for "good"), label "1" in Diabetes indicates the presence of diabetes (the abnormal case), label "M" in Sonar indicates the presence of a mine (instead of a normal rock), and label "M" in WDBC indicates the sampled breast mass is malignant. In these datasets, the label that is the straightforward choice for self performs much better than when using the "abnormal" samples as self. This shows that choice of self is important in KERNSA, but, as one would expect, the reasoning behind the choice of self does not need to be sophisticated.

4.4 External Comparisons

The top performing classifiers from each dataset are used to calculate the accuracy as discussed in Section 3.2.2. These accuracies are presented in Table 4.24 with the best accuracy for each dataset highlighted. For datasets with only two labels, the label used as self to calculate the accuracy is shown in parenthesis.

The top accuracy from each dataset in Table 4.24 is compared against accuracies from other AIS algorithms in Table 4.25 with the winning AIS algorithm bolded. Note that KERNSA ranks first in only one of the six datasets, but its accuracy is usually within 5% of the accuracy of the top algorithm. Also, recall that the parameter sweep in these experiments optimizes on AUC, not accuracy; if accuracy had been used as the performance value in the parameter sweep, these values may have been higher. Unfortunately, AUCs are generally not published, or a more direct comparison could be made on AUCs.

Table 4.24: Accuracies by dataset and affinity function. If the dataset has two classes, the label treated trained as self is shown.

	Linear	Gaussian	I. Multiq.	Sigmoid
Diabetes	75.36% (0)	71.39% (0)	**75.50% (0)**	70.36% (0)
Ionosphere	68.93% (g)	72.38% (g)	72.20% (g)	**82.48% (g)**
Iris	95.31%	**95.67%**	79.93%	91.73%
Sonar	**85.27% (R)**	80.10% (R)	82.40% (R)	81.38% (M)
WDBC	92.13% (B)	92.20% (B)	**93.57% (B)**	93.51% (B)
Wine	64.93%	92.05%	**94.87%**	85.34%

4.5 Summary

This chapter presents and analyzes the results of the experiments described in Chapter 3. The values found from the parameter sweep and used in the experiments are given. Tests of statistical significance are performed on the results to compare them to the Linear affinity function. Some interesting findings in the results are remarked upon. This chapter concludes by comparing KERNSA's accuracies on these datasets to other AIS learners.

Table 4.25: Accuracy comparison to other algorithms.

	KERNSA	AIRS1	AIRS2	AISLFS	Immunos	M-NSA	MINSA	Parallel AIRS
Reference		[9]	[9]	[24]	[10]	[55]	[55]	[9]
Diabetes	**75.50**	69.70	70.87	74.21	-	70.44	72.00	70.96
Ionosphere	82.48	87.04	85.13	**94.37**	-	-	-	84.44
Iris	95.67	95.27	95.00	95.71	**97.33**	95.33	96.00	95.60
Sonar	85.27	69.81	66.01	**88.10**	68.51	-	-	64.71
WDBC	93.57	**96.84**	96.17	96.42	86.14	96.37	96.51	96.47
Wine	94.87	-	-	**97.76**	-	-	-	-

V. Conclusions and Recommendations

While a new Machine Learning (ML) algorithm will never outperform existing algorithms in all problem domains [40, 84], this research shows that the Kernel Extended Real-valued Negative Selection Algorithm (KERNSA) offers performance improvements over a comparable Real-valued Negative Selection Algorithm (RNSA) implementation. The Sigmoid KERNSA seems particularly well suited to a subset of the test domains.

KERNSA is compared to other Artificial Immune System (AIS) algorithms and found to perform well, though it only ranks first in accuracy in one of the six datasets compared. Since the KERNSA classifiers used in this research are optimized for Area Under Curve (AUC) instead of accuracy, optimizing directly for accuracy may yield better results in this comparison but is left to future work.

This research also demonstrates that continuous output from RNSA detectors is possible and the method used here can be mapped directly back to the original RNSA decision boundary. Continuous output is used in this research to generate Receiver Operating Characteristic (ROC) curves and calculate AUCs, but can also be used as a basis of classification confidence or probability.

5.1 Future Work

5.1.1 Placement of Supporting Vectors.

KERNSA places each supporting vector in the input space $\mathcal{X}$ randomly. Two improvements over random placement are suggested: adopting more of the V-detector functionality and/or placing supporting vectors according to some pattern or heuristic.

More V-detector-like functionality. The idea of setting the radius of each detector as the distance to the closest self sample comes from the V-detector algorithm, but the V-detector algorithm does not stop there. The V-detector also iteratively moves detectors

away from the closest self sample to get larger radii and further generalize the classifier [16]. As an additional measure, a detector that is within a certain distance of self samples or a new detector that is centered within an existing detector is rejected in the V-detector algorithm [16]. Similarly, KERNSA could move supporting vectors away from nearby self samples and reject supporting vectors that are too close to self samples or are contained within existing detectors.

Patterns of placement. The shape of the resulting detection area is determined by the location of the supporting vectors in $\mathcal{X}$. By controlling the pattern of vector placement such that the position of the next supporting vector depends on the locations of the detector's previous supporting vectors, the approximate shape of the detector can be pre-determined. For example, the "detection shape" formed in Figure 2.8b closely matches the shape of the vectors used in the distance calculation for that shape.

5.1.2 Self Buffer and Detector Radius.

In many RNSA implementations, the detection area maintains some minimum distance from the self samples, creating a buffer zone between the self samples and the detectors [16]. This is done so that the classifier can generalize, since it is likely that the immediate area around a self sample will also belong to future self samples. Implementing this in KERNSA would be difficult due to the space transformation (distances differ between affinities and directions change), but it may offer significant improvements. Note that this solution is an absolute buffer area, not a relative one; a relative buffer is already possible in KERNSA by setting radii of detectors to some percentage of the distance to the nearest self sample.

A closely related, easier to implement improvement is controlling the radius of the detectors by rejecting detectors that do not meet a minimum radius. This minimum radius would need to vary depending on the affinity function and the affinity parameters. Alternatively, more detectors could be generated than necessary and those with the shortest

radii dropped. While these solutions do not create a buffer around the self samples, it still helps generalize the generated classifier.

5.1.3 Determine Which Affinity Performs Best With Each type of Dataset.

As seen in the results from the experiments, the Sigmoid affinity function performs decidedly better than the other affinities on some of the dataset-label pairs. Future work could analyze the datasets Sigmoid performs better on in order to make recommendations on what conditions using the Sigmoid affinity function is probably the best choice. Though Sigmoid had the sharpest performance differences in the experiments, similar analysis could be conducted for all affinities in order to make recommendations for those as well.

5.1.4 More Kernel Functions.

This research used four of the most popular kernel functions as the affinities, but there are many more available which may prove useful for some problem domains. Combining kernel functions to form a new kernel function is also an area of future research for KERNSA.

5.1.5 Automate the Parameter Sweep.

The parameter sweeping in this research is an entirely manual process with a set level of granularity. If this process were automated, searching a larger area of parameters at finer detail without a large computational increase may be possible, which would allow further tuning to increase the performance of KERNSA.

5.1.6 Tune To Accuracy for More Direct Comparisons.

This research used accuracy to compare an AUC-tuned KERNSA to other AIS algorithms. Tuning for accuracy would offer a more direct comparison to these other algorithms and give a clearer picture of each algorithm's general performance.

5.1.7 Handling of Misclassified Samples.

When training a one-class classifier, an abundance of samples may be available, but the labels may not be known. For example, this could be the case when sampling an engine's

hourly operation. Several weeks of sampling may be available, but it may be uncertain if the engine was operating abnormally during that time. Future work for KERNSA could research handling samples that were used during training that have been misclassified.

5.2 Summary

This chapter reviews the work and findings of this thesis. Future work for KERNSA is also discussed.

Bibliography

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[2] Stephen D. Bay, Dennis Kibler, Michael J. Pazzani, and Padhraic Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *SIGKDD Explorations Newsletter*, 2(2):81–85, December 2000.

[3] Richard Bellman, Richard Ernest Bellman, and Richard Ernest Bellman. *Adaptive control processes: A guided tour*. Princeton University Press, 1966.

[4] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999.

[5] P. Berkhin. A survey of clustering data mining techniques. In Jacob Kogan, Charles Nicholas, and Marc Teboulle, editors, *Grouping Multidimensional Data*, pages 25–71. Springer Berlin Heidelberg, 2006.

[6] Jason M Bindewald. Detector design considerations in high-dimensional artificial immune systems. Master's thesis, The Air Force Institute of Technology, 2012.

[7] Chris M Bishop. Novelty detection and neural network validation. In *Vision, Image and Signal Processing, IEE Proceedings*, volume 141, pages 217–222. IET, May 1994.

[8] Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

[9] Jason Brownlee. Artificial immune recognition system (AIRS)-a review and analysis. Technical Report 1-02, Centre for Intelligent Systems and Complex Processes, January 2005.

[10] Jason Brownlee. Immunos-81, the misunderstood artificial immune system. Technical Report 1-03, Centre for Intelligent Systems and Complex Processes, January 2005.

[11] Radha Chitta, Rong Jin, Timothy C. Havens, and Anil K. Jain. Approximate kernel k-means: solution to large scale kernel clustering. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 895–903, New York, NY, USA, 2011. ACM.

[12] Andrzej Chmielewski and Slawomir T Wierzchon. V-detector algorithm with tree-based structures. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 9–14, 2006.

[13] Carlos A Coello Coello, Daniel Cortes Rivera, and Nareli Cruz Cortes. Use of an artificial immune system for job shop scheduling. In *Artificial Immune Systems*, pages 1–10. Springer Berlin Heidelberg, 2003.

[14] D. Dasgupta. Advances in artificial immune systems. *Computational Intelligence Magazine, IEEE*, 1(4):40–49, 2006.

[15] Dipankar Dasgupta. *Artificial Immune Ssytems: Datasets*, 2013. URL http://ais.cs.memphis.edu/index.php?page=datasets.

[16] Dipankar Dasgupta and Fernando Nino. *Immunological computation: theory and applications*. Auerbach Publications, 2008.

[17] Leandro N De Castro and Fernando J Von Zuben. Learning and optimization using the clonal selection principle. *Evolutionary Computation, IEEE Transactions on*, 6(3): 239–251, 2002.

[18] Leandro Nunes De Castro and Jonathan Timmis. *Artificial immune systems: a new computational intelligence approach*. Springer, 2002.

[19] Leandro Nunes de Castro and Fernando J Von Zuben. *aiNet: an artificial immune network for data analysis*, chapter XII, pages 231–259. Idea Group Publishing, USA, 2001.

[20] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[21] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556. ACM, 2004.

[22] T.G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[23] Thomas Dietterich. Ensemble methods in machine learning. *Multiple classifier systems*, pages 1–15, 2000.

[24] Grzegorz Dudek. An artificial immune system for classification with local feature selection. *Evolutionary Computation, IEEE Transactions on*, 2012.

[25] Stephanie Forrest, Alan S Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 202–212. Ieee, 1994.

[26] Alex Freitas and Jon Timmis. Revisiting the foundations of artificial immune systems: A problem-oriented perspective. *Artificial Immune Systems*, pages 229–241, 2003.

[27] Alex A Freitas and Jon Timmis. Revisiting the foundations of artificial immune systems for data mining. *Evolutionary Computation, IEEE Transactions on*, 11(4): 521–540, 2007.

[28] Jerome H Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1(1):55–77, 1997.

[29] Maoguo Gong, Licheng Jiao, Lining Zhang, and Haifeng Du. Immune secondary response and clonal selection inspired optimizers. *Progress in Natural Science*, 19(2): 237–253, 2009.

[30] Maoguo Gong, Jian Zhang, Jingjing Ma, and Licheng Jiao. An efficient negative selection algorithm with further training for anomaly detection. *Knowledge-Based Systems*, 2012.

[31] Fabio A González and Dipankar Dasgupta. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403, 2003.

[32] Guodong Guo, Stan Z Li, and Kapluk Chan. Face recognition by support vector machines. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 196–201. IEEE, 2000.

[33] T Guzella, T Mota-Santos, and W Caminhas. Artificial immune systems and kernel methods. *Artificial Immune Systems*, pages 303–315, 2008.

[34] D.J. Hand. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine learning*, 77(1):103–123, 2009.

[35] Steven A Hofmeyr and Stephanie Forrest. Architecture for an artificial immune system. *Evolutionary computation*, 8(4):443–473, 2000.

[36] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[37] K Igawa and H Ohashi. A negative selection algorithm for classification and reduction of the noise effect. *Applied Soft Computing*, 9(1):431–438, 2009.

[38] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[39] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[40] N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

[41] Nathalie Japkowicz. *Concept-learning in the absence of counter-examples: an autoassociation-based approach to classification*. PhD thesis, Rutgers, The State University of New Jersey, 1999.

[42] Zhou Ji and Dipankar Dasgupta. Real-valued negative selection algorithm with variable-sized detectors. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 287–298. Springer, 2004.

[43] Zhou Ji and Dipankar Dasgupta. Applicability issues of the real-valued negative selection algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 111–118. ACM, 2006.

[44] Zhou Ji and Dipankar Dasgupta. V-detector: An efficient negative selection algorithm with "probably adequate" detector coverage. *Information sciences*, 179(10):1390–1406, 2009.

[45] Zeng Jinquan, Liu Xiaojie, Li Tao, Liu Caiming, Peng Lingxi, and Sun Feixian. A self-adaptive negative selection algorithm used for anomaly detection. *Progress in Natural Science*, 19(2):261–266, 2009.

[46] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

[47] Dae-Won Kim, Ki Young Lee, Doheon Lee, and Kwang H Lee. Evaluation of the performance of clustering algorithms in kernel-induced feature space. *Pattern Recognition*, 38(4):607–611, 2005.

[48] Jungwon Kim, Peter J Bentley, Uwe Aickelin, Julie Greensmith, Gianni Tedesco, and Jamie Twycross. Immune system approaches to intrusion detection–a review. *Natural computing*, 6(4):413–466, 2007.

[49] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Lawrence Erlbaum Associates Ltd, 1995.

[50] Doudou LaLoudouana, Mambobo Bonouliqui Tarare, Lupano Tecallonou Center, and GUANA Selacie. Data set selection. *Journal of Machine Learning Gossip*, 1:11–19, 2003.

[51] Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods, 2003.

[52] Charles Ling, Jin Huang, and Harry Zhang. AUC: a better measure than accuracy in comparing learning algorithms. *Advances in Artificial Intelligence*, 2003.

[53] Charles X Ling, Jin Huang, and Harry Zhang. AUC: a statistically consistent and more discriminating measure than accuracy. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 519–526. Lawrence Erlbaum Associates Ltd, 2003.

[54] J.M. Lobo, A. Jiménez-Valverde, and R. Real. AUC: a misleading measure of the performance of predictive distribution models. *Global Ecology and Biogeography*, 17 (2):145–151, 2007.

[55] Urszula Markowska-Kaczmar and Bartosz Kordas. Multi-class iteratively refined negative selection classifier. *Applied Soft Computing*, 8(2):972–984, 2008.

[56] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. ISBN 0070428077.

[57] Tom M Mitchell. Machine learning and data mining. *Communications of the ACM*, 42 (11):30–36, 1999.

[58] Mary M Moya and Don R Hush. Network constraints and multi-objective optimization for one-class classification. *Neural Networks*, 9(3):463–474, 1996.

[59] K.R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, 12(2): 181–201, 2001.

[60] Marek Ostaszewski, Franciszek Seredynski, and Pascal Bouvry. Coevolutionary-based mechanisms for network anomaly detection. *Mathematical Modeling and Algorithms*, 2007.

[61] Seral Özşen, S Gunes, Sadık Kara, and Fatma Latifoglu. Use of kernel functions in artificial immune systems for the nonlinear classification problems. *Information Technology in Biomedicine, IEEE Transactions on*, 13(4):621–628, 2009.

[62] Patrick Pantel, Dekang Lin, et al. Spamcop: A spam classification & organization program. In *Proceedings of AAAI-98 Workshop on Learning for Text Categorization*, pages 95–98, 1998.

[63] Alan S Perelson and George F Oster. Theoretical studies of clonal selection: minimal antibody repertoire size and reliability of self-non-self discrimination. *Journal of theoretical biology*, 81(4):645–670, 1979.

[64] Jeff M. Phillips and Suresh Venkatasubramanian. A gentle introduction to the kernel distance. *CoRR*, abs/1103.1625, 2011.

[65] Dorian Pyle. *Data preparation for data mining*, volume 1. Morgan Kaufmann, 1999.

[66] Jie Qin, Darrin P Lewis, and William Stafford Noble. Kernel hierarchical gene clustering from microarray expression data. *Bioinformatics*, 19(16):2097–2104, 2003.

[67] Gunter Ritter and María Teresa Gallegos. Outliers in statistical pattern recognition and an application to automatic chromosome classification. *Pattern Recognition Letters*, 18(6):525–539, 1997.

[68] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, Upper Saddle River, NJ, third edition, 2010.

[69] Raúl E Sánchez-Yáñez, Evguenii V Kurmyshev, and Antonio Fernández. One-class texture classifier in the CCR feature space. *Pattern Recognition Letters*, 24(9):1503–1511, 2003.

[70] Bernhard Schölkopf. The kernel trick for distances. *Advances in neural information processing systems*, pages 301–307, 2001.

[71] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[72] Joseph M Shapiro, Gary B Lamont, and Gilbert L Peterson. An evolutionary algorithm to generate hyper-ellipsoid detectors for negative selection. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 337–344. ACM, 2005.

[73] Chris Sinclair, Lyn Pierce, and Sara Matzner. An application of machine learning to network intrusion detection. In *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*, pages 371–377. IEEE, 1999.

[74] Thomas Stibor and Jonathan Timmis. Comments on real-valued negative selection vs. real-valued positive selection and one-class svm. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 3727–3734, 2007.

[75] Thomas Stibor, Philipp Mohr, Jonathan Timmis, and Claudia Eckert. Is negative selection appropriate for anomaly detection? In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 321–328. ACM, 2005.

[76] Thomas Stibor, Jonathan Timmis, and Claudia Eckert. A comparative study of real-valued negative selection to statistical anomaly detection techniques. *Artificial Immune Systems*, pages 262–275, 2005.

[77] Thomas Stibor, Jonathan Timmis, and Claudia Eckert. On the use of hyperspheres in artificial immune systems as antibody recognition regions. *Artificial Immune Systems*, pages 215–228, 2006.

[78] David Tax and Robert Duin. Data domain description using support vectors. In *Proceedings of the European Symposium on Artificial Neural Networks*, volume 256. Citeseer, 1999.

[79] David Martinus Johannes Tax. *One-class classification: concept-learning in the absence of counter-examples*. PhD thesis, Delft University of Technology, June 2001.

[80] David MJ Tax and Robert PW Duin. Feature scaling in support vector data descriptions. Technical report, American Association for Artificial Intelligence, 2000.

[81] Jon Timmis, Mark Neal, and John Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1):143–150, 2000.

[82] Andrew Watkins, Jon Timmis, and Lois Boggess. Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, 2004.

[83] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, third edition, 2011.

[84] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[85] Peifei WU and Xufei ZHENG. An improved variable-radius real-valued negative selection algorithm. *Journal of Information and Computational Science*, 2012.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
13–06–2013	Master's Thesis	Oct 2011–June 2013

4. TITLE AND SUBTITLE

Kernel Extended Real-Valued Negative Selection Algorithm (KERNSA)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Smith, Brett Andrew, Civ

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-13-J-07

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Artificial Immune Systems (AISs) are a type of statistical Machine Learning (ML) algorithm based on the Biological Immune System (BIS) applied to classification problems. Inspired by increased performance in other ML algorithms when combined with kernel methods, this research explores using kernel methods as the distance measure for a specific AIS algorithm, the Real-valued Negative Selection Algorithm (RNSA). This research also demonstrates that the hard binary decision from the traditional RNSA can be relaxed to a continuous output, while maintaining the ability to map back to the original RNSA decision boundary if necessary. Continuous output is used in this research to generate Receiver Operating Characteristic (ROC) curves and calculate Area Under Curves (AUCs), but can also be used as a basis of classification confidence or probability. The resulting Kernel Extended Real-valued Negative Selection Algorithm (KERNSA) offers performance improvements over a comparable RNSA implementation. Using the Sigmoid kernel in KERNSA seems particularly well suited (in terms of performance) to four out of the eighteen domains tested.

15. SUBJECT TERMS

Artificial Intelligence, Artificial Immune Systems, Negative Selection Algorithms, Kernel Methods

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gilbert Peterson
U	U	U	UU	87	19b. TELEPHONE NUMBER *(include area code)* (937) 255-6565 ext. 4281 gilbert.peterson@afit.edu